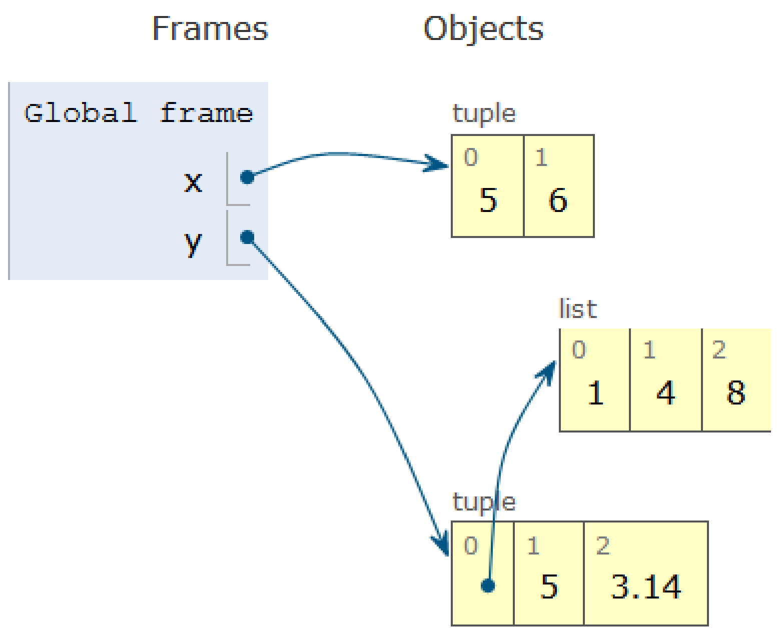
Compsci 101
DeMorgan's Law, Short circuiting, Global, Tuples



Susan Rodger

February 23, 2023

L is for …

- **Loops**
  - While, For, Nested – Iteration!
- **Library**
  - Where we find APIs and Implementations
- **Logic**
  - Boolean expressions in if statements, loops
- **Linux**
  - The OS that runs the world?

Keith Kirkland

- **BS ME, BFA Accessories Design, MID Industrial and Product Design**
- **Co-founder of WearWorks**
- **Wayband – wearable haptic navigation device for blind**
- **Device guided blind marathon runner in NYC marathon**

“We design products that shift people’s lives in a meaningful way”

“We take large challenges and turn them into opportunities that will one day help people and awaken the problems that can be solved. We believe in setting new standards for what is possible. ”

Announcements

- **APT-3 due tonight**
- **Assign 3 due Thursday, March 2**
  - Sakai Assign 3 quiz due Tues. Feb 28 (no grace day!)
- **Lab 6 on Friday, do prelab**
- **Midterm grades coming – rough estimate!**

- **APT Quiz 1 – Feb 23 (today 1pm) – Mon, Feb 27**

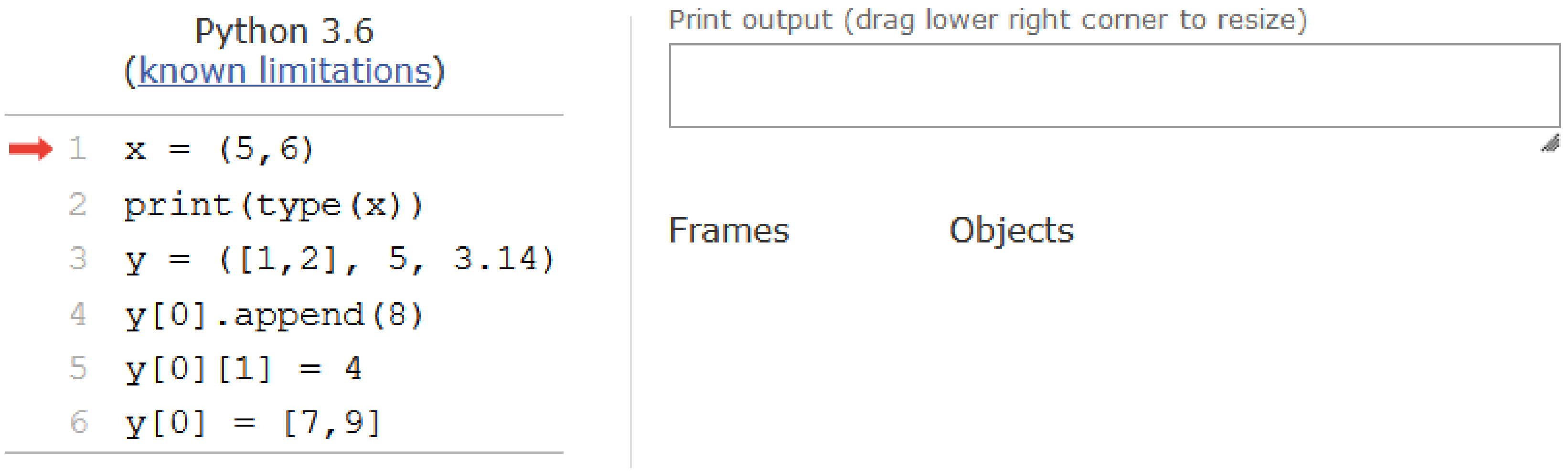
PFTD

- **Tuples**
- **Global**
- **DeMorgan’s Law**
- **Short Circuiting**
- **APT Quiz**

Tuple: What and Why?

- **Similar to a list in indexing starting at 0**
  - Can store any type of element
  - Can iterate over
- **Immutable - Cannot mutate/change its value(s)**
  - Efficient because it can't be altered
- **Examples:**
  - **`x = (5,6)`**
  - **`y = ([1,2],3.14)`**

Tuple Trace in Python Tutor



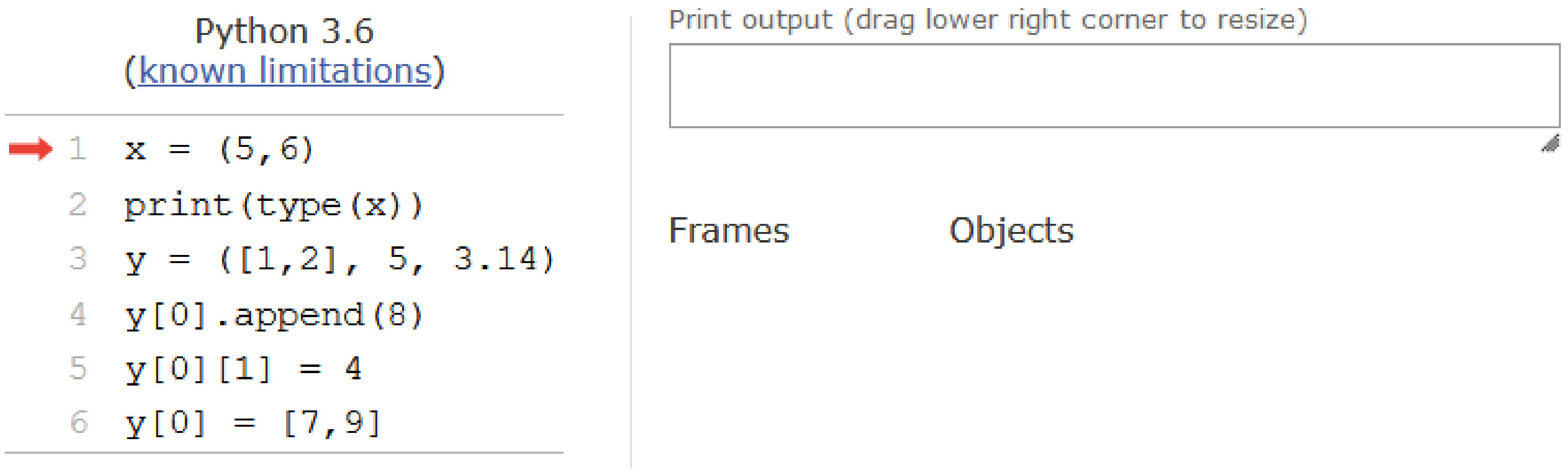
Python 3.6
(known limitations)

```
x = (5,6)
print(type(x))
y = ([1,2], 5, 3.14)
y[0].append(8)
y[0][1] = 4
y[0] = [7,9]
```

Print output (drag lower right corner to resize)

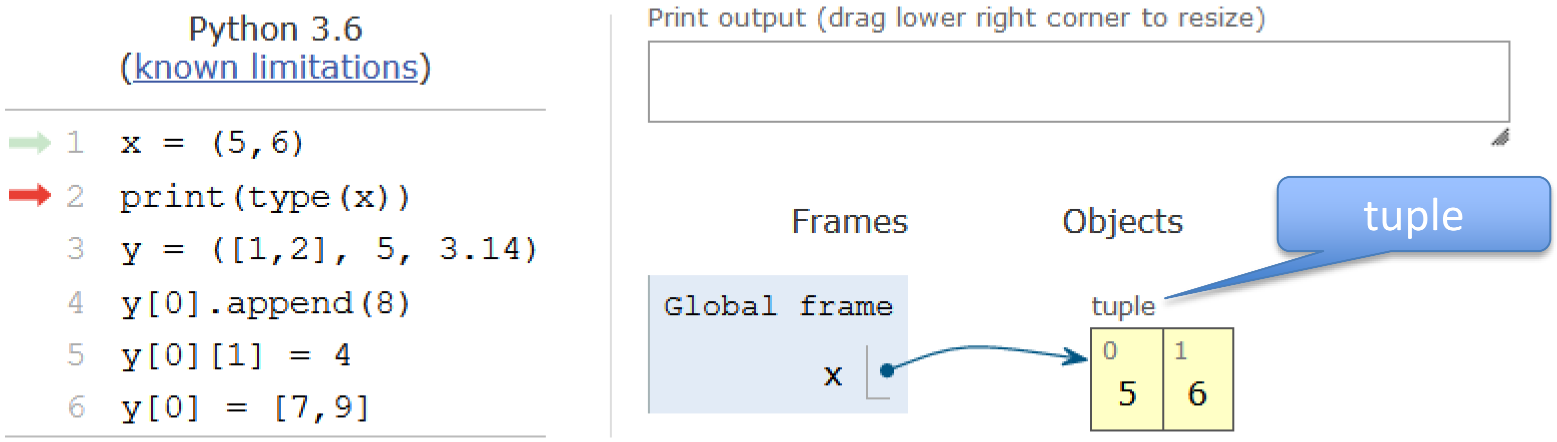
Frames Objects

Tuple Trace in Python Tutor



Python 3.6
(known limitations)

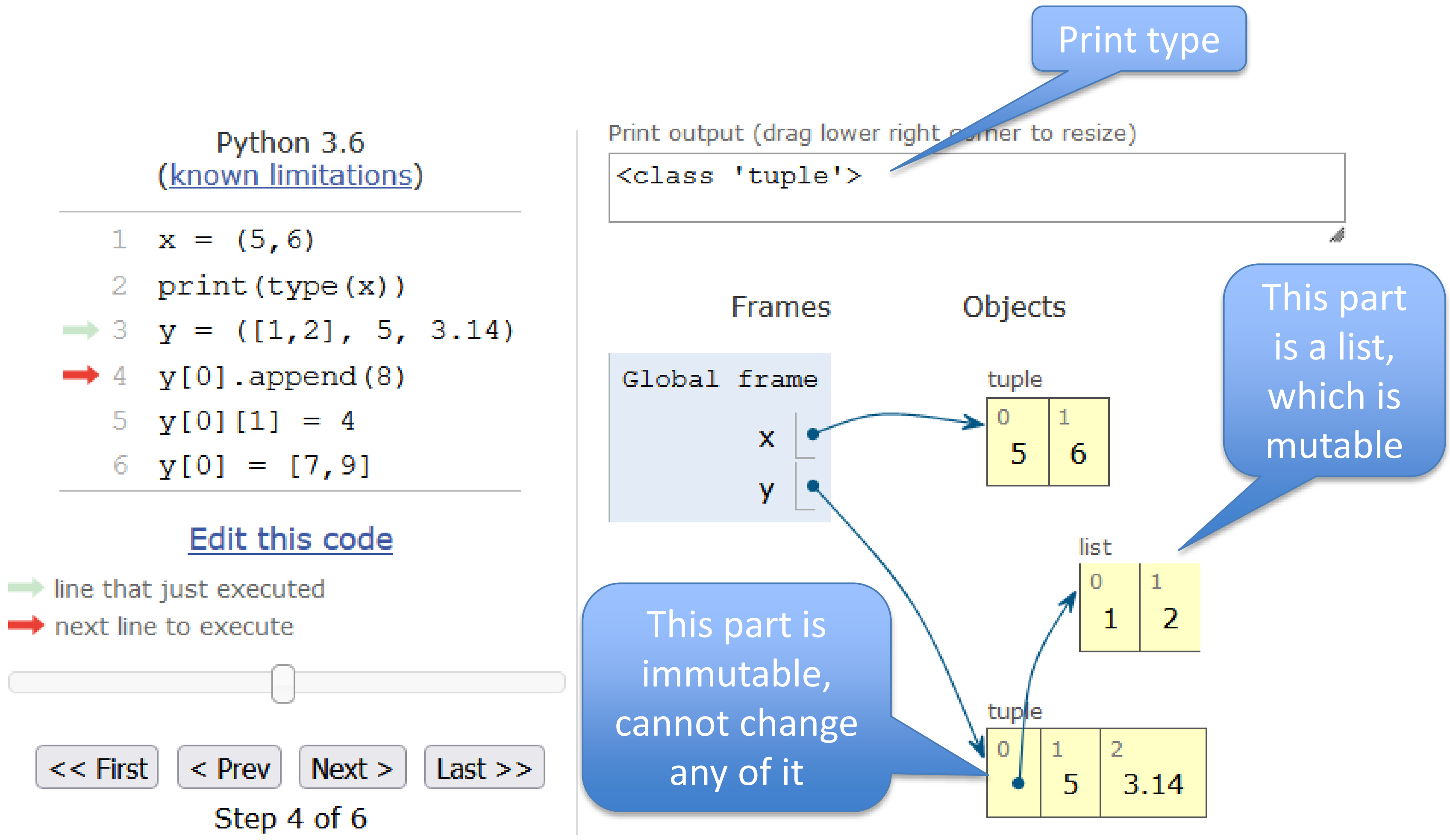
```
x = (5,6)
print(type(x))
y = ([1,2], 5, 3.14)
y[0].append(8)
y[0][1] = 4
y[0] = [7,9]
```



Python 3.6
(known limitations)

```
x = (5,6)
print(type(x))
y = ([1,2], 5, 3.14)
y[0].append(8)
y[0][1] = 4
y[0] = [7,9]
```

Tuple Trace in Python Tutor


Python 3.6
(known limitations)

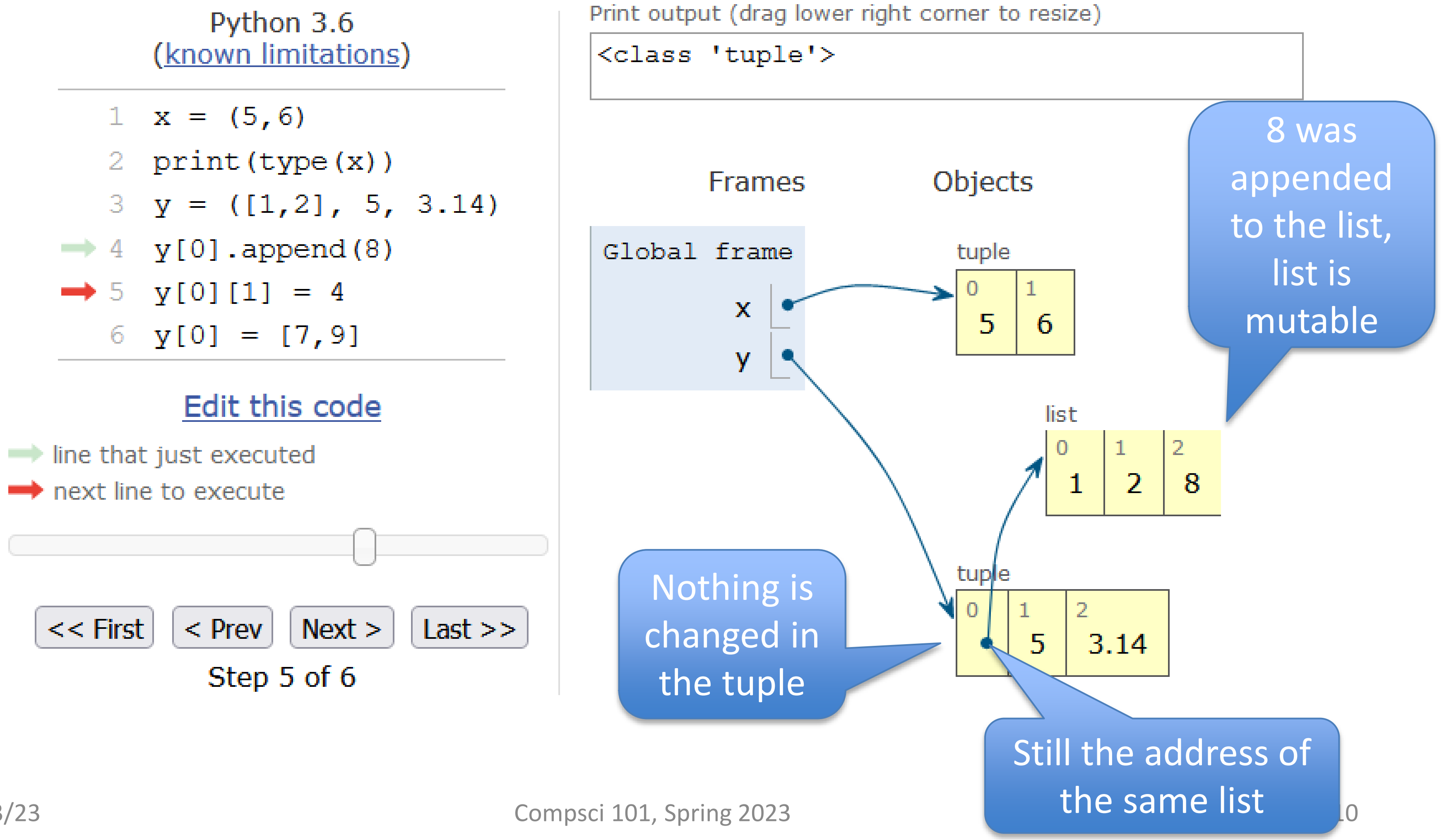
```
x = (5,6)
print(type(x))
y = ([1,2], 5, 3.14)
y[0].append(8)
y[0][1] = 4
y[0] = [7,9]
```

Edit this code

Print output (drag lower right corner to resize)

```
<class 'tuple'>
```

Tuple Trace in Python Tutor

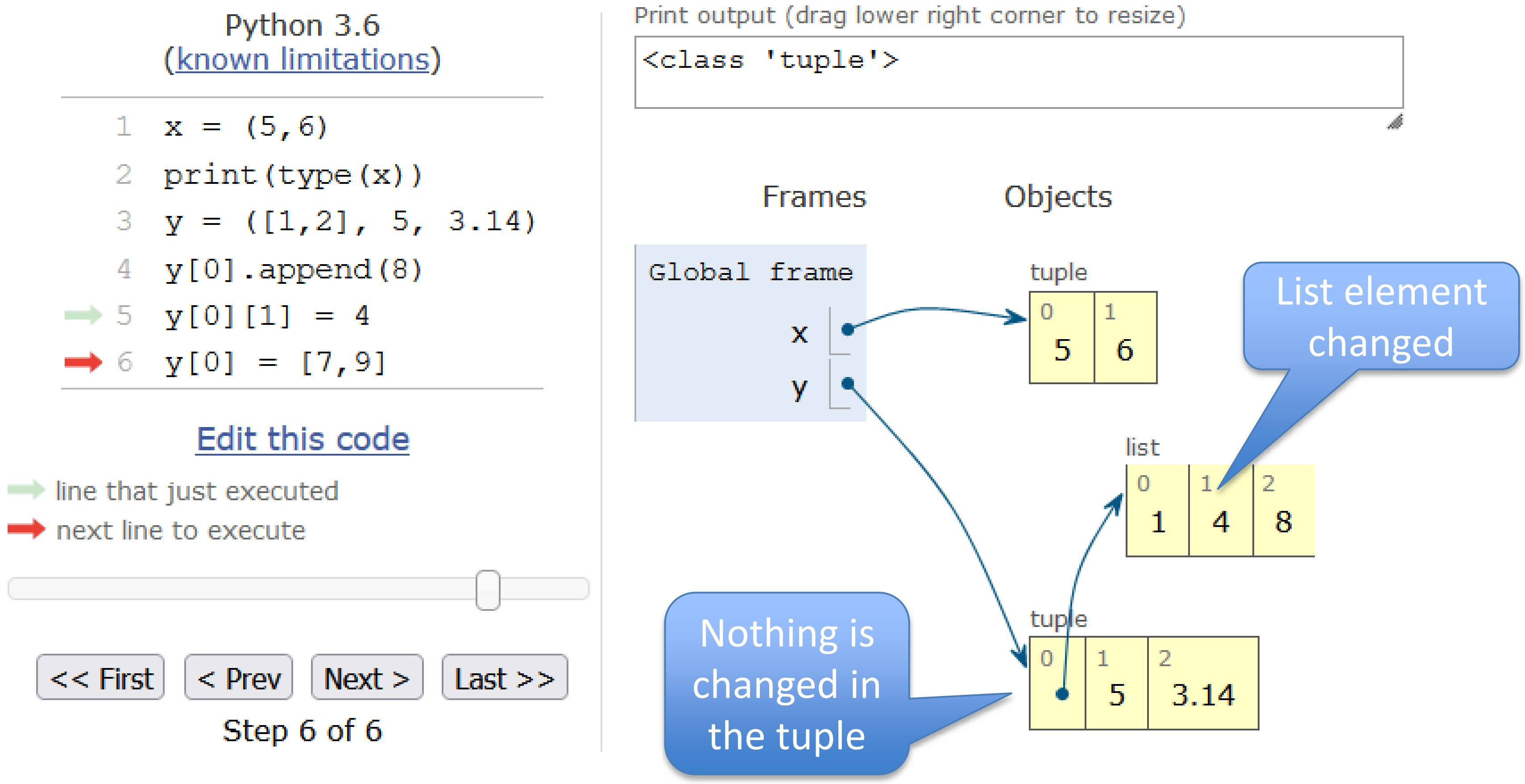


```
x = (5,6)
print(type(x))
y = ([1,2], 5, 3.14)
y[0].append(8)
y[0][1] = 4
y[0] = [7,9]
```

Print output (drag lower right corner to resize)

```
<class 'tuple'>
```

Tuple Trace in Python Tutor



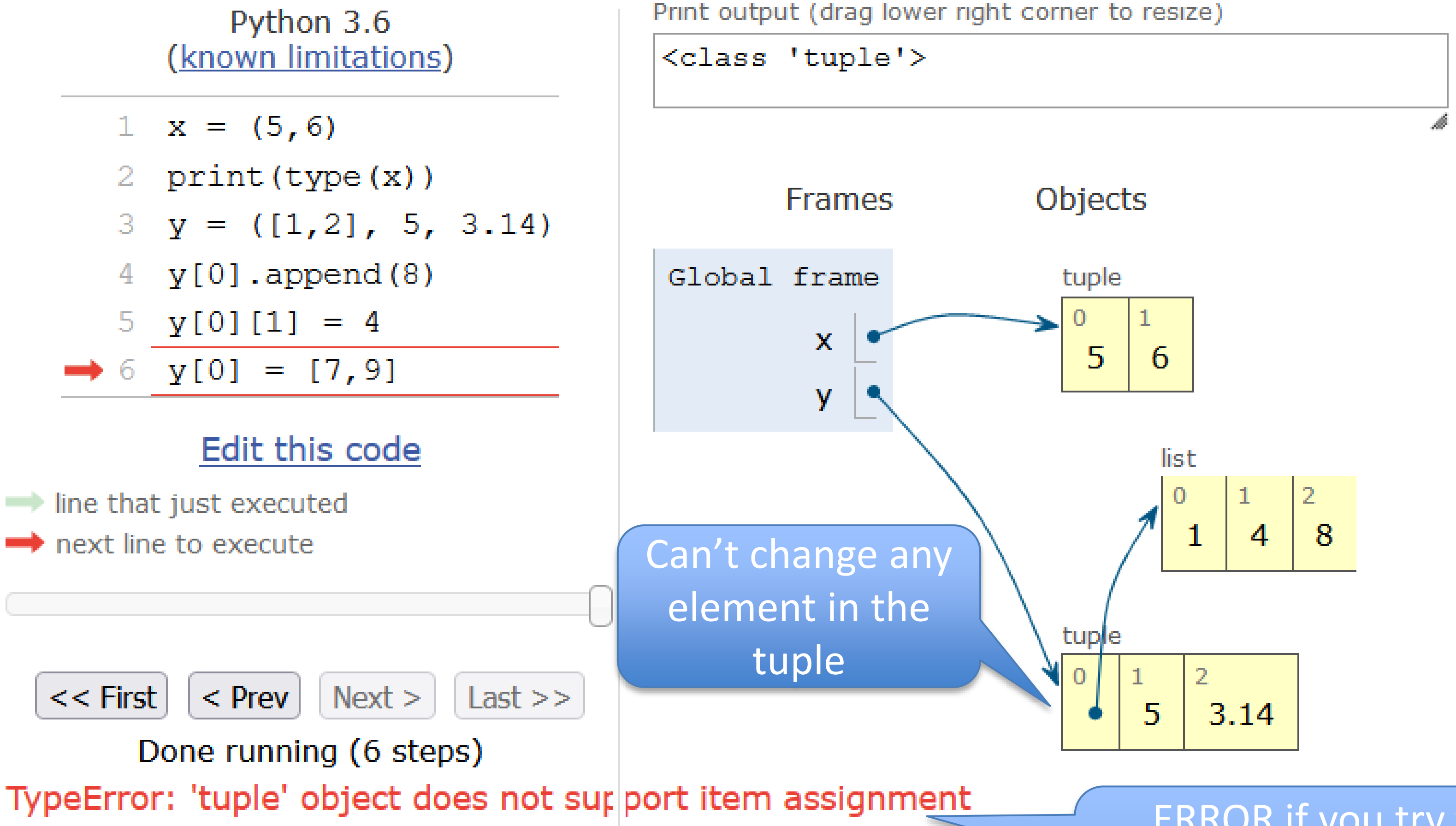
Python 3.6
(known limitations)

```
x = (5,6)
print(type(x))
y = ([1,2], 5, 3.14)
y[0].append(8)
y[0][1] = 4
y[0] = [7,9]
```

Print output (drag lower right corner to resize)

```
<class 'tuple'>
```

Tuple Trace in Python Tutor



Python 3.6
(known limitations)

```
x = (5,6)
print(type(x))
y = ([1,2], 5, 3.14)
y[0].append(8)
y[0][1] = 4
y[0] = [7,9]
```

Print output (drag lower right corner to resize)

```
<class 'tuple'>
```

Done running (6 steps)

TypeError: 'tuple' object does not support item assignment

Can't change any element in the tuple

ERROR if you try to change any part of the tuple

Variables and their Scope

- **Local variable – variable in function only known in that function**

- **Parameter – way to pass information to a function**

- **Global variable - variable known throughout the whole file**

What is a global variable?

- **Accessible everywhere in the file (or “module”)**
- **Variable is in the global frame**
  - First frame in Python Tutor
- **If declared global in a function:**
  - The variable in the global frame can also be reassigned in that function
  - Despite Python being in a different frame!
- **Eliminates the need to pass this value to all the functions that need it**

When to use Global Variables

- **Typically, don’t use global variables**
  - Harder to share a function if it refers to a global variable
  - Act differently than other variables

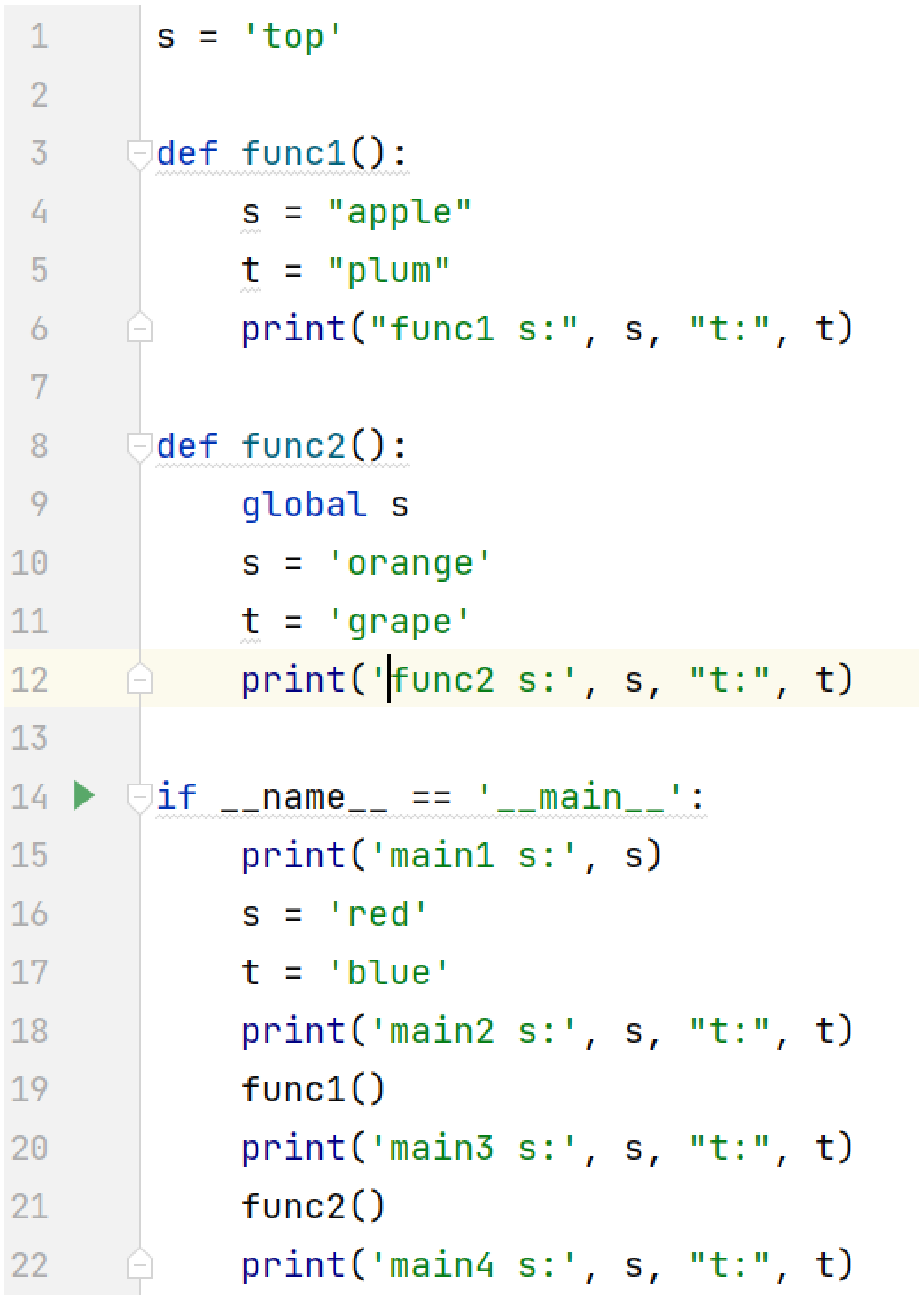
- **Sometimes makes sense**
  - Global variable is used in most functions
  - Saves passing it to every function

- **Best practice = help other humans read the code**
  - Global variables define at top of file
  - When global used in function, declared as global at beginning of function

When reading code with globals

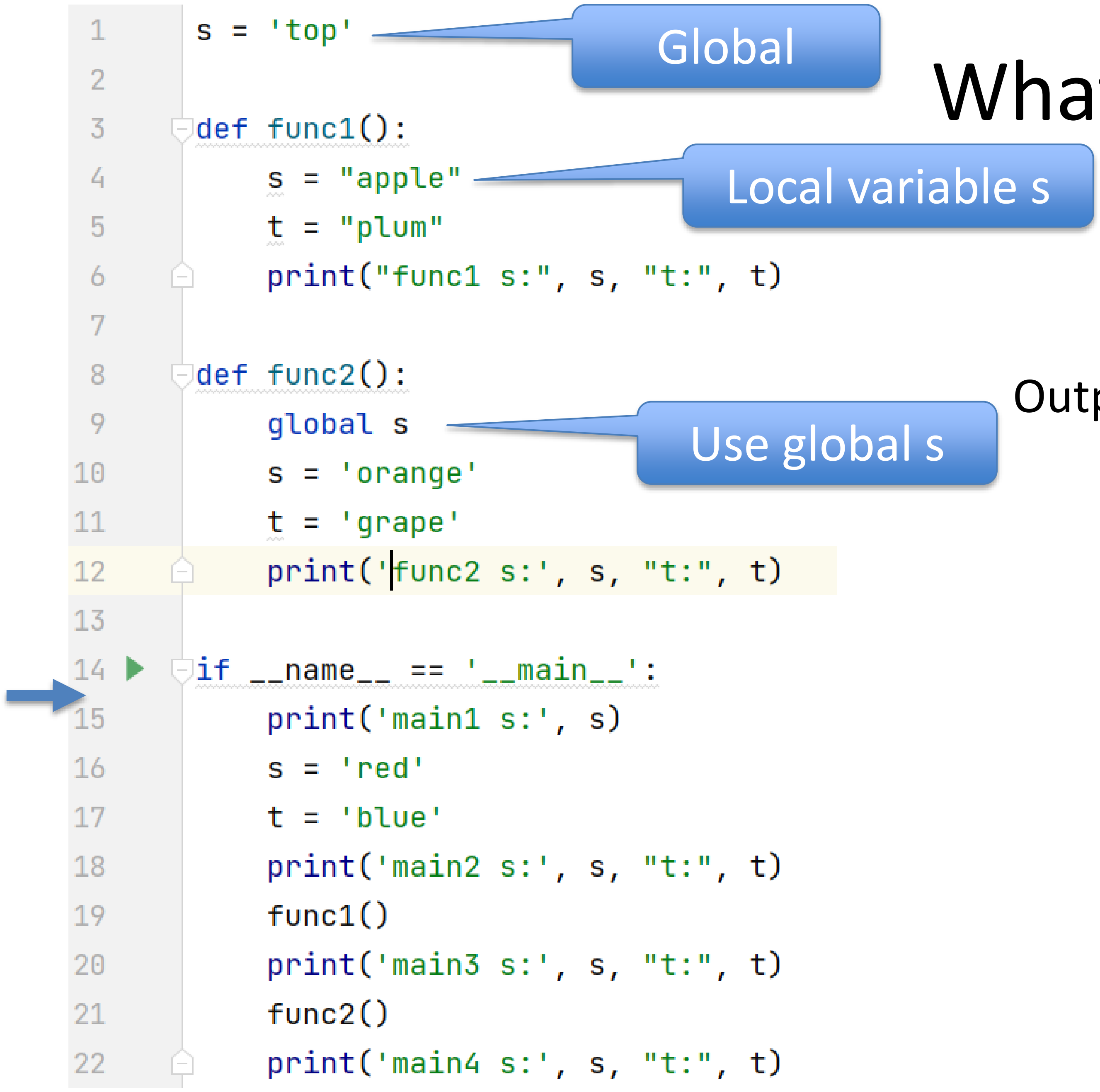
- **When checking the value of a variable, ask:**
  - Is this variable local to the function or in the global frame?
- **When in a function and assigning a value to a variable, ask:**
  - Has this variable been declared global?
    - If yes, reassign the variable in the **global frame**
    - If no, create/reassign the variable in the **function's local frame**

What will print?



```python
s = 'top'

def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

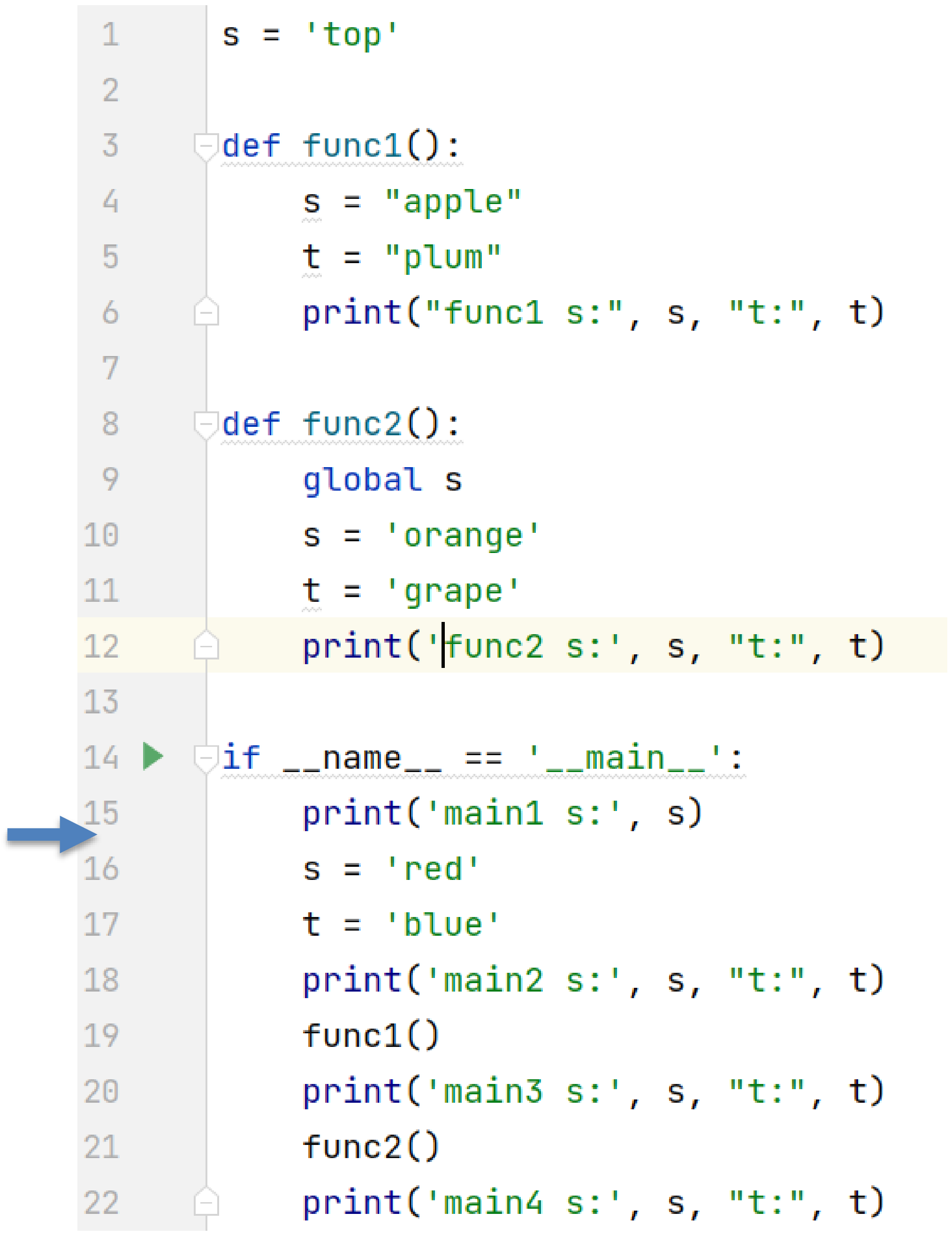
What will print?



```python
s = 'top'

def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

Output:

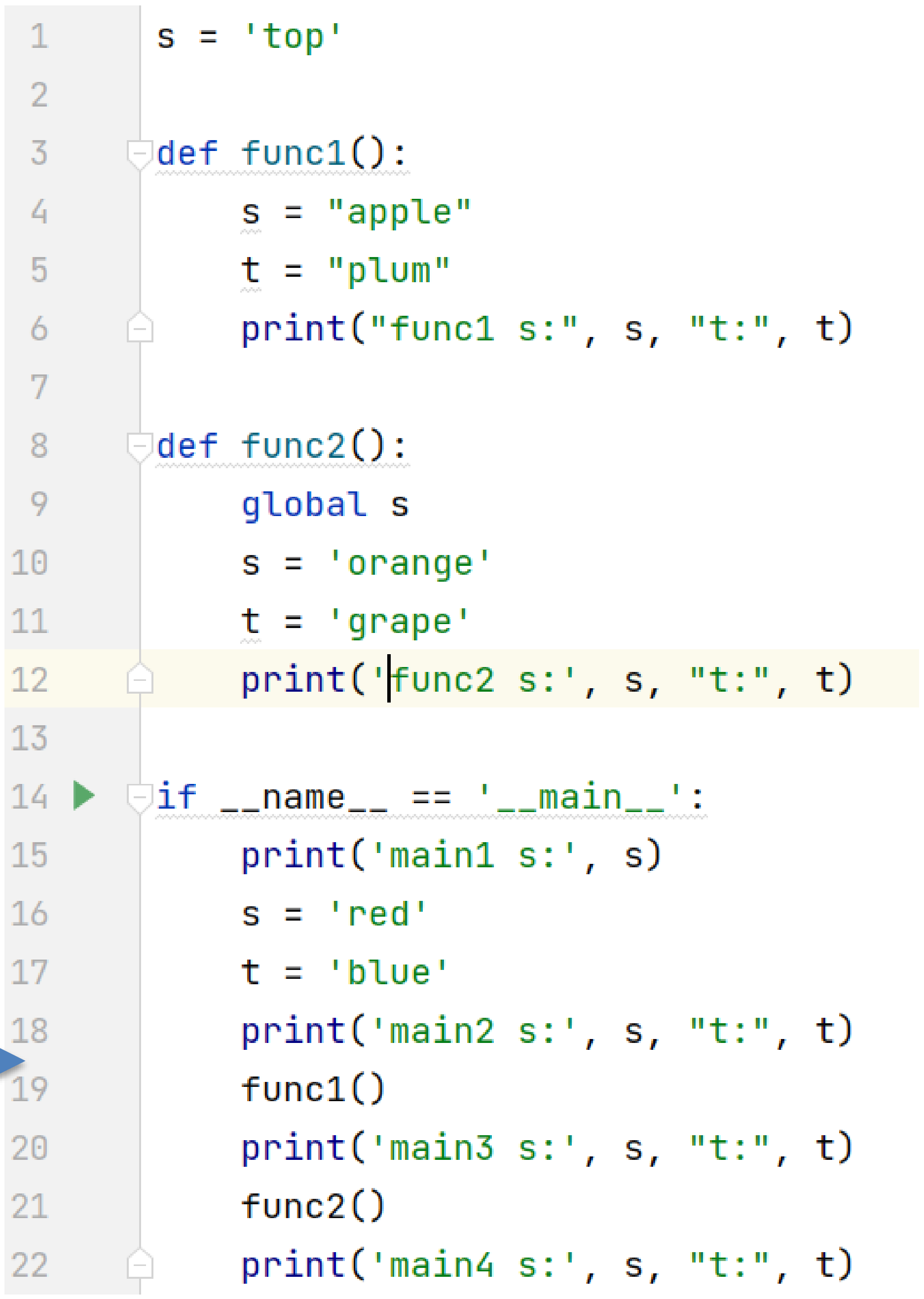
What will print?



```python
s = 'top'

def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

Output:

```
main1 s: top
```



```python
s = 'top'

def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

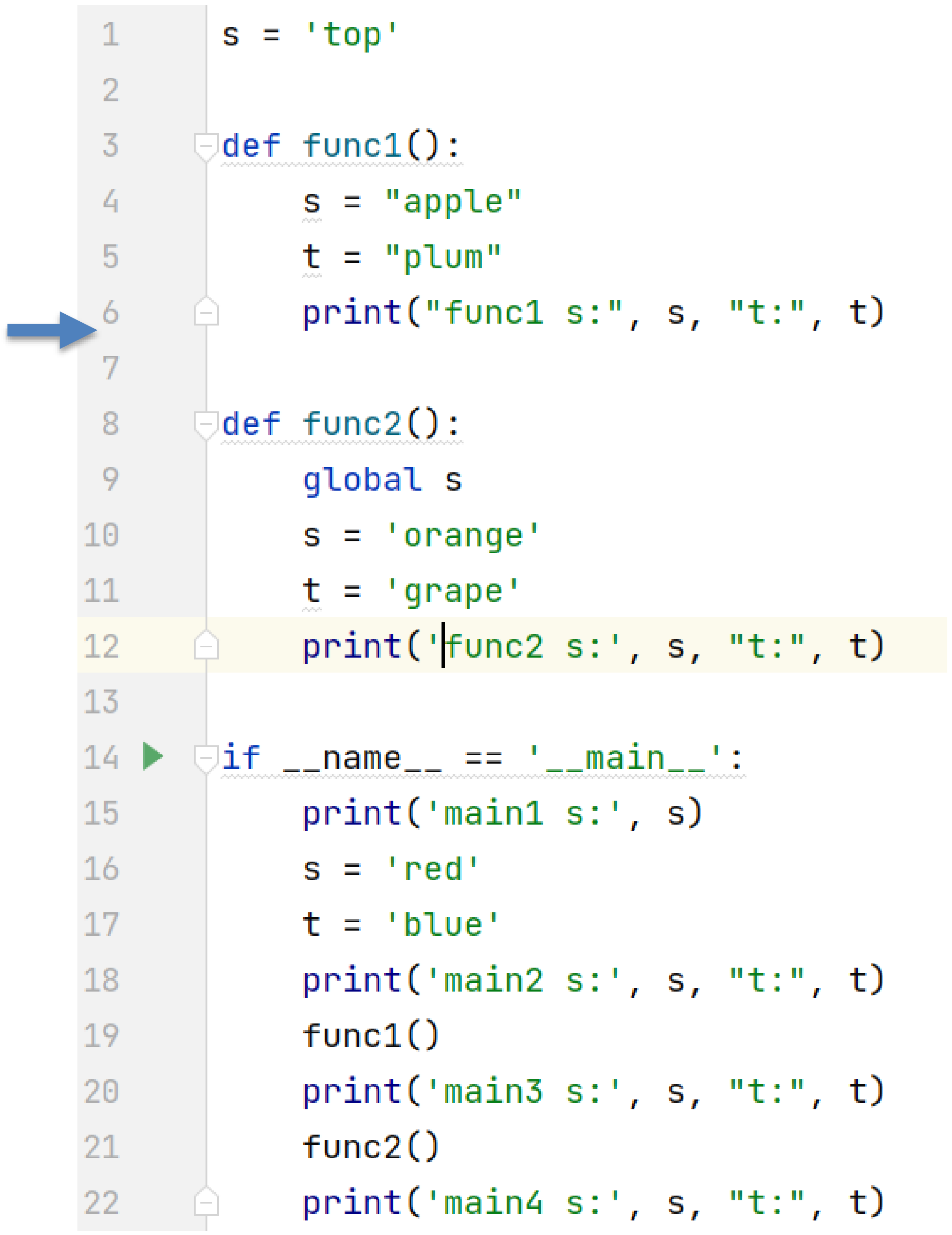
What will print?

Output:

main1 s: top

main2 s: red t: blue

Next call func1

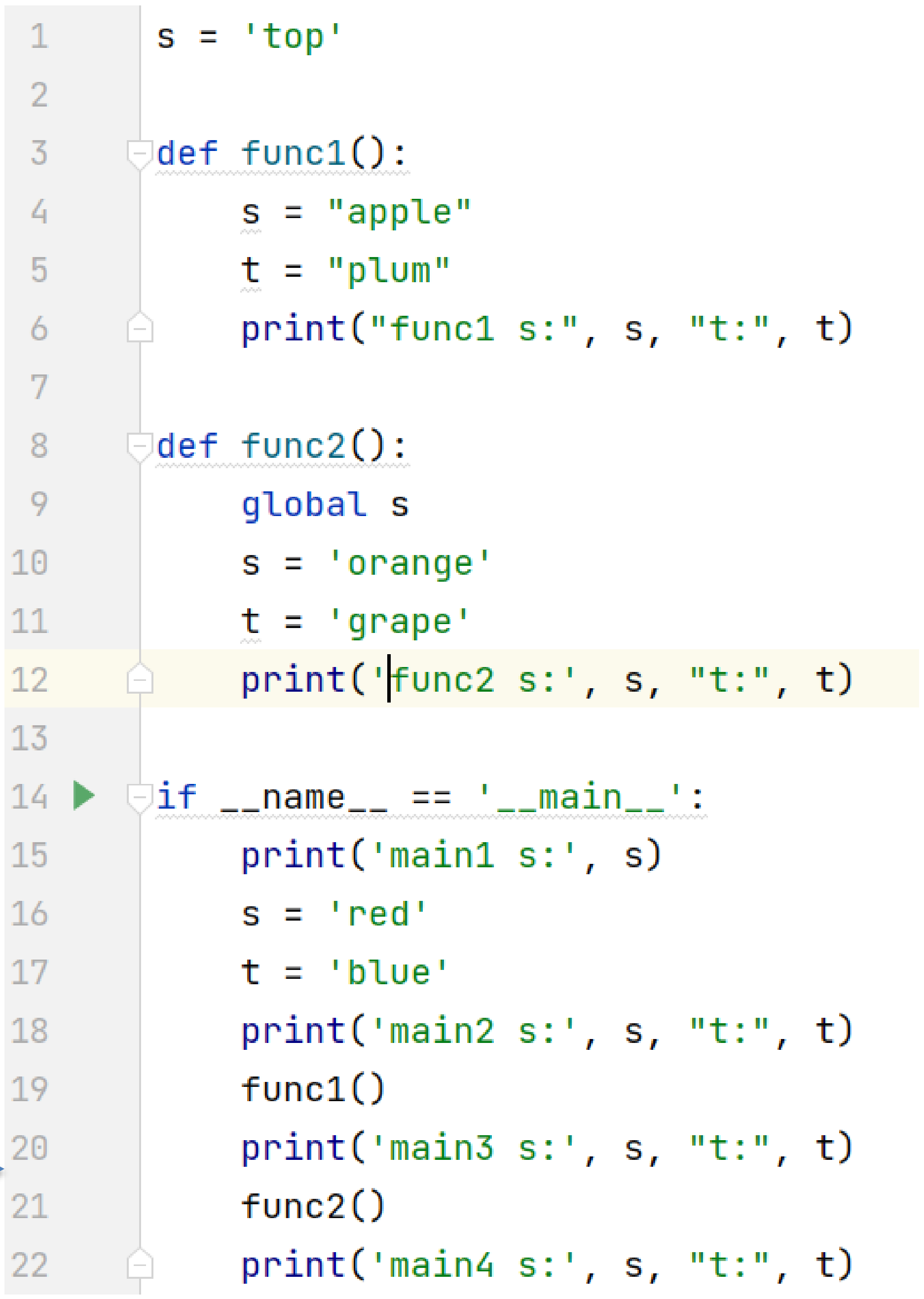


```python
s = 'top'

def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

What will print?

Output:

```
main1 s: top
main2 s: red t: blue
func1 s: apple t: plum
```



```python
s = 'top'

def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

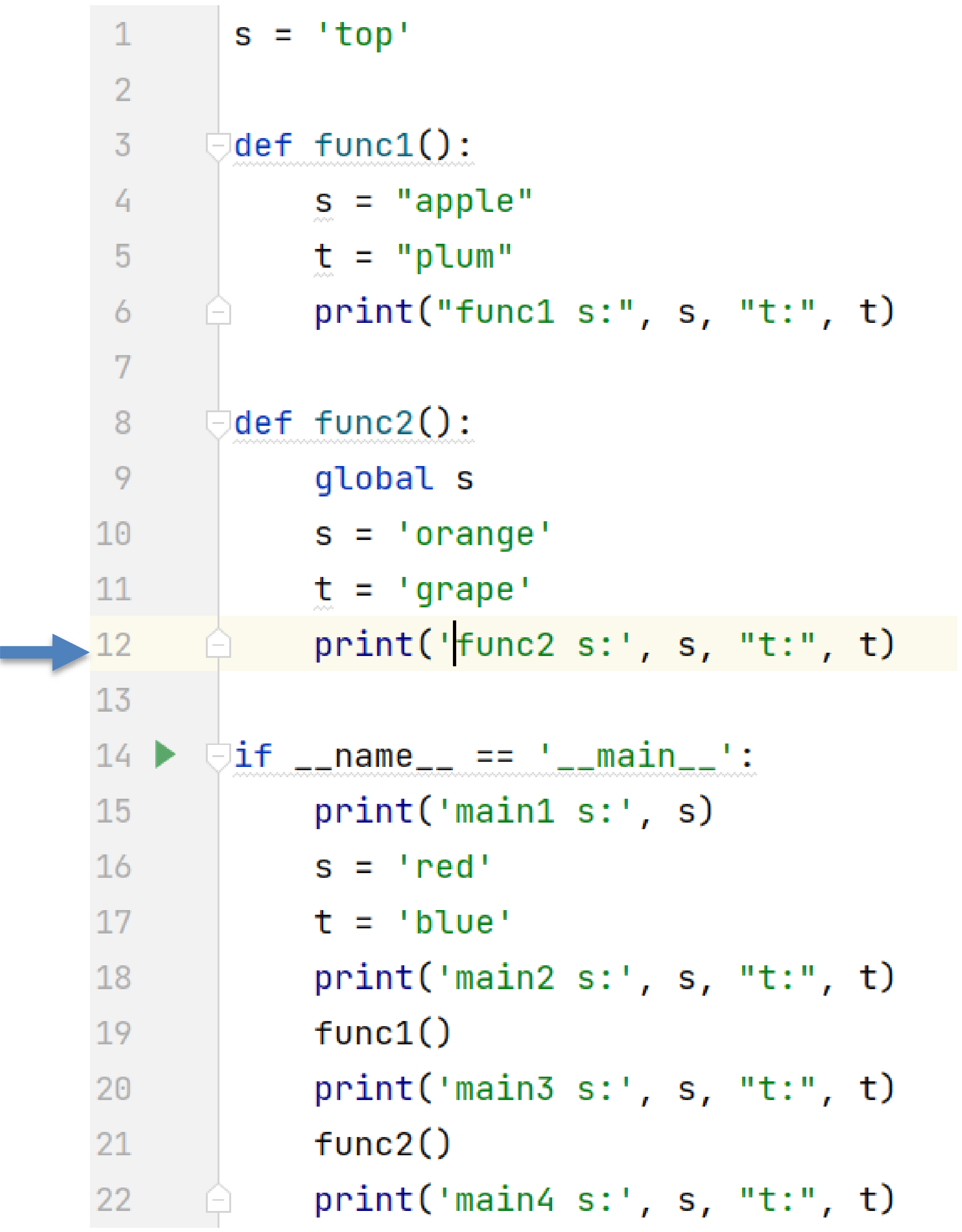
What will print?

Output:

```
main1 s: top
main2 s: red t: blue
func1 s: apple t: plum
main3 s: red t: blue
```

Next call func2

What will print?

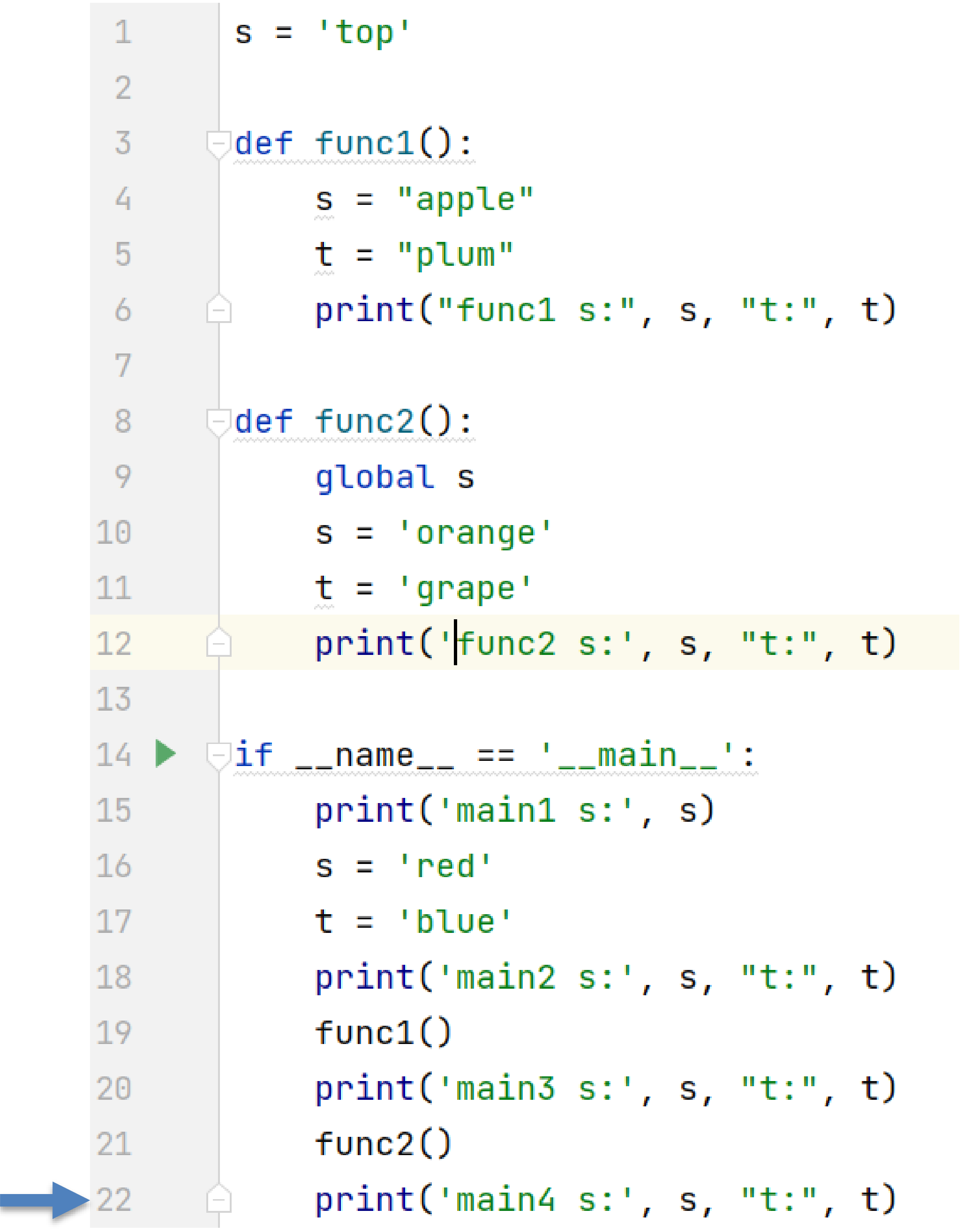


```python
s = 'top'

def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

Output:

```
main1 s: top
main2 s: red t: blue
func1 s: apple t: plum
main3 s: red t: blue
func2 s: orange t: grape
```

What will print?

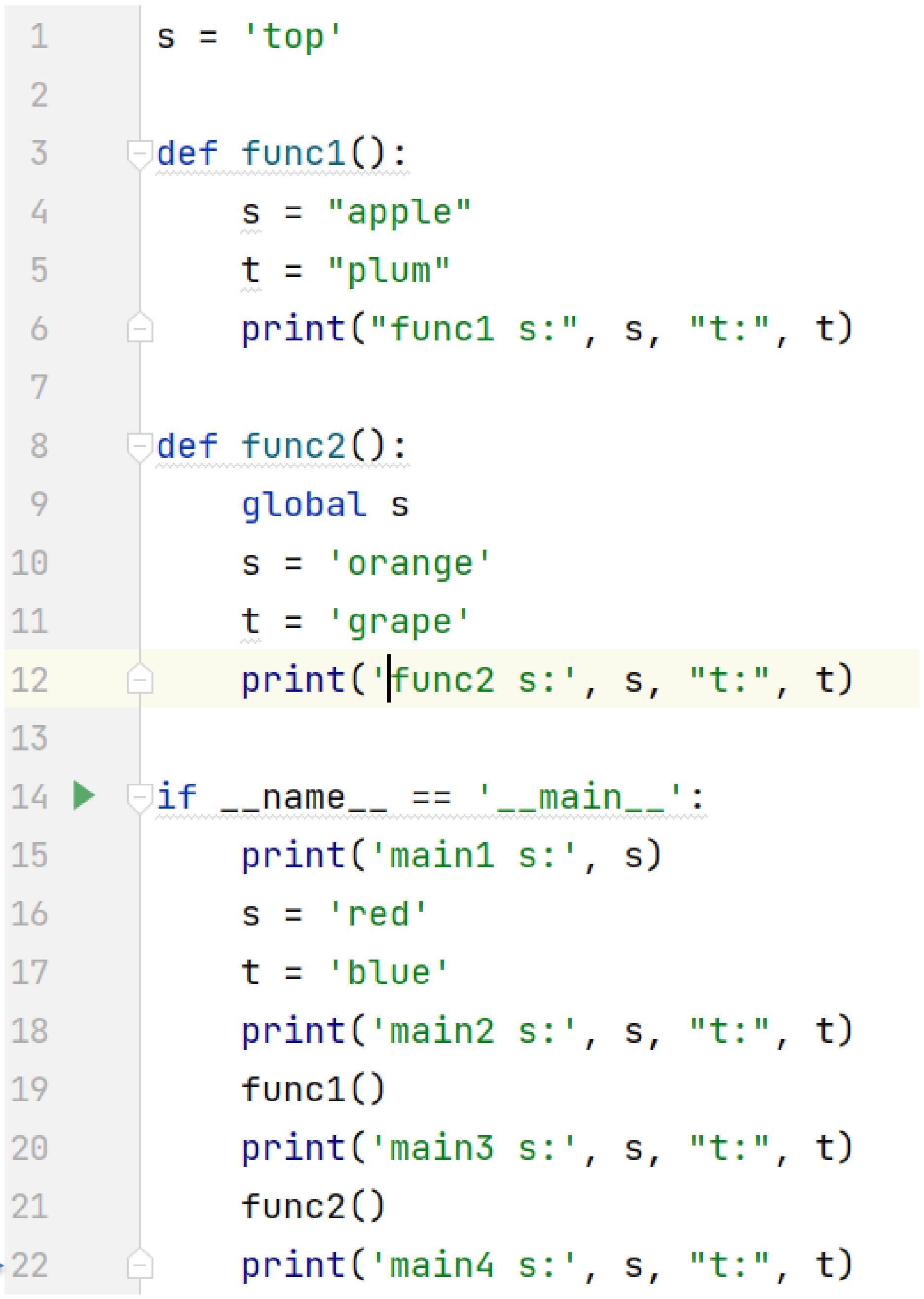


```python
s = 'top'

def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

Output:

```
main1 s: top
main2 s: red t: blue
func1 s: apple t: plum
main3 s: red t: blue
func2 s: orange t: grape
main4 s: orange t: blue
```

What will print?



```python
s = 'top'

def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

Output:

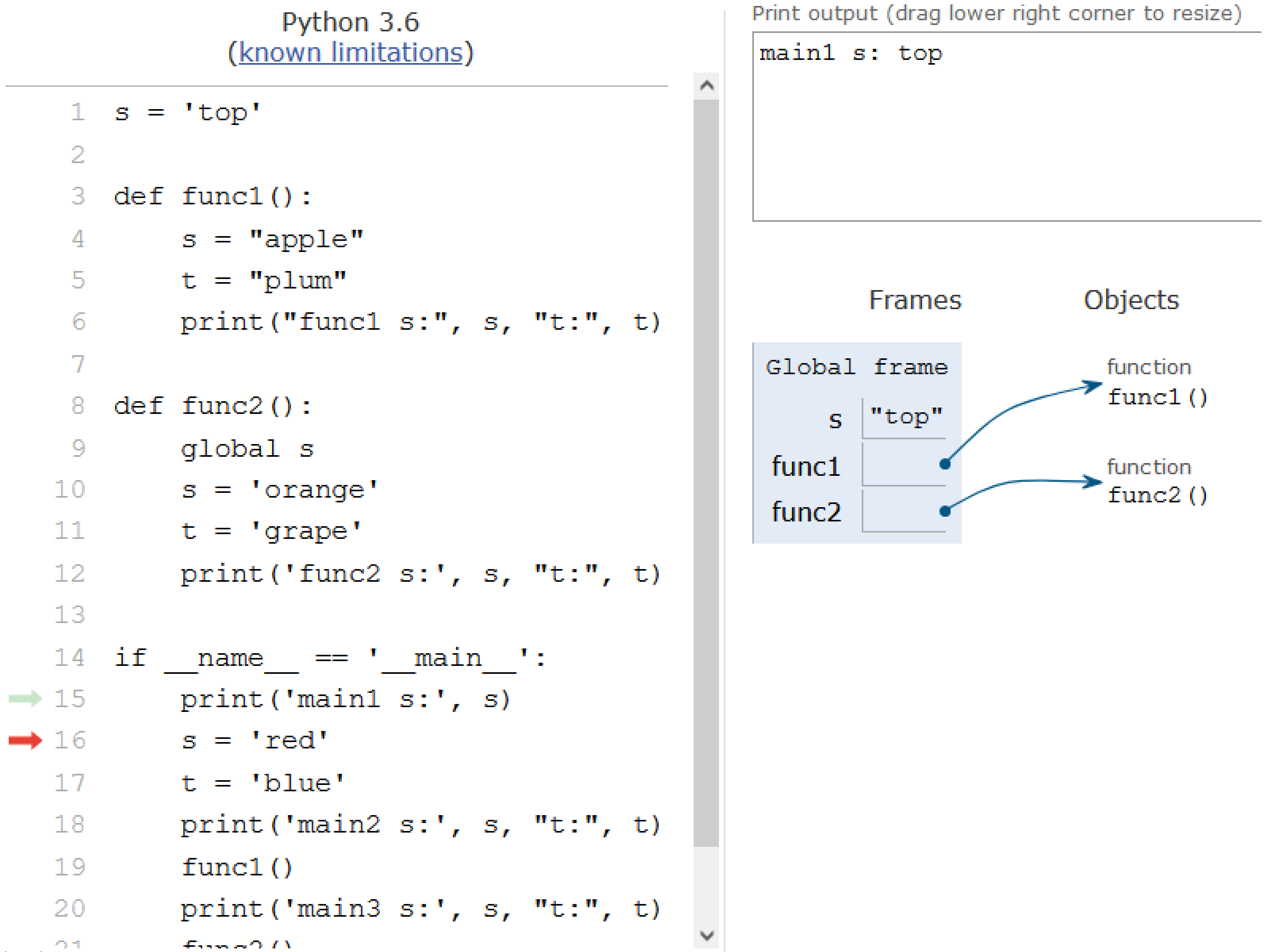
```
main1 s: top
main2 s: red t: blue
func1 s: apple t: plum
main3 s: red t: blue
func2 s: orange t: grape
main4 s: orange t: blue
```

Notice t in main is always "blue"
s in main changed to "orange"

Now let’s see the same thing in Python Tutor

- **Global variables are in the global frame**

Python Tutor – Step 6



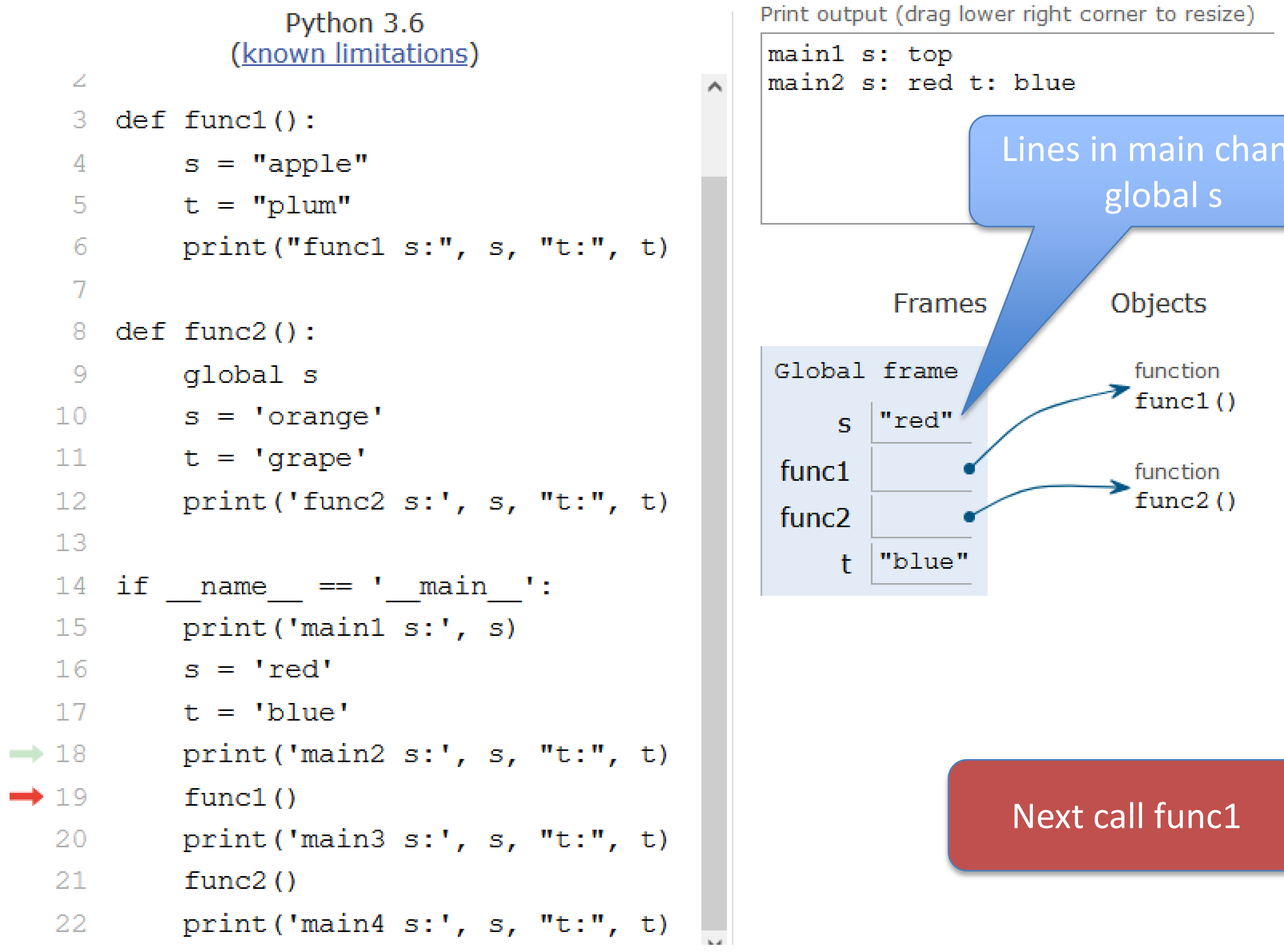
Python 3.6
(known limitations)

```
s = 'top'

def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
```

Print output (drag lower right corner to resize)

```
main1 s: top
```

Python Tutor – Step 9



Python 3.6
(known limitations)

```
def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

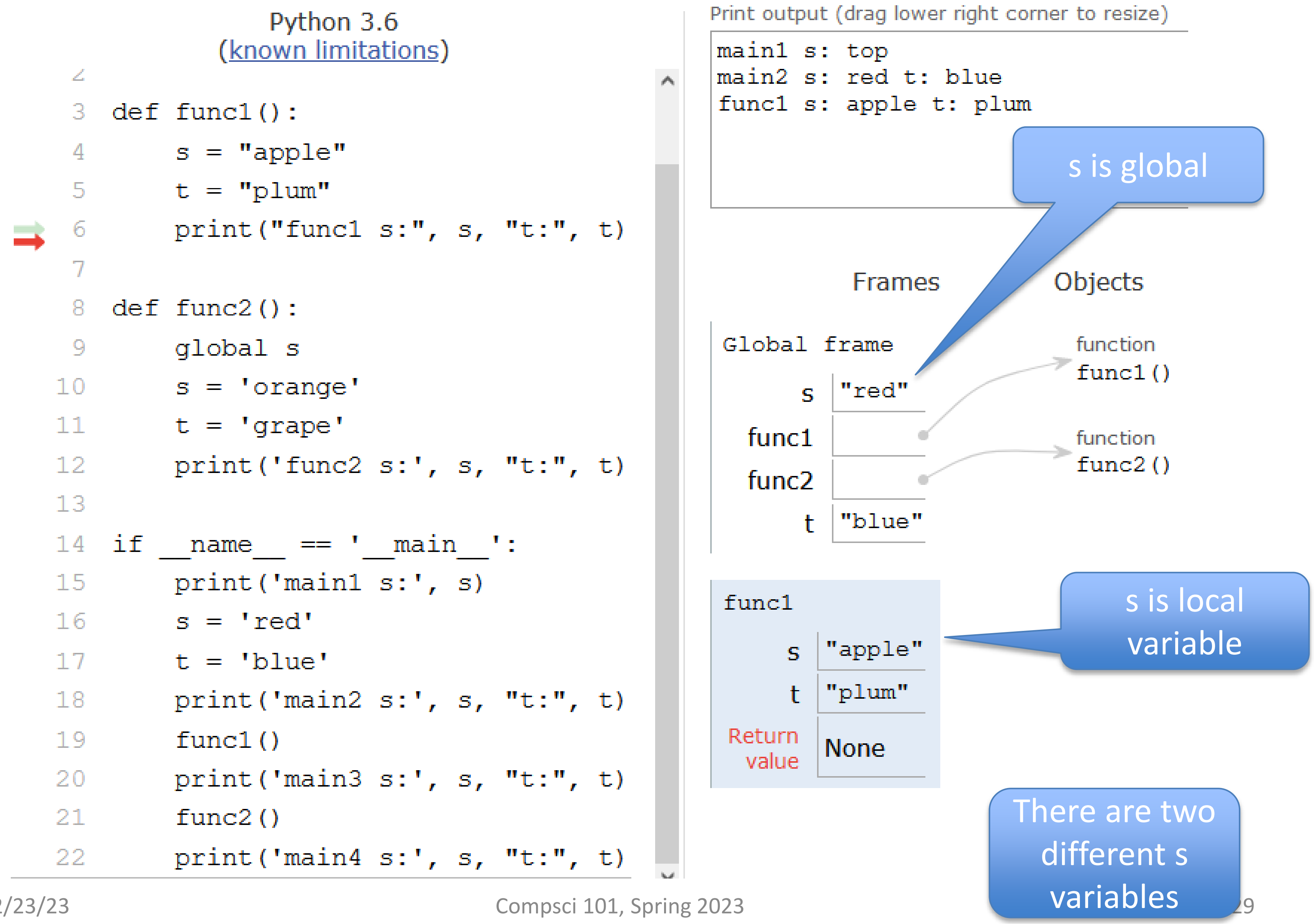
Print output (drag lower right corner to resize)

```
main1 s: top
main2 s: red t: blue
```

Lines in main change global s

Next call func1

Python Tutor – Step 14



Python 3.6
(known limitations)

```
def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

Print output (drag lower right corner to resize)

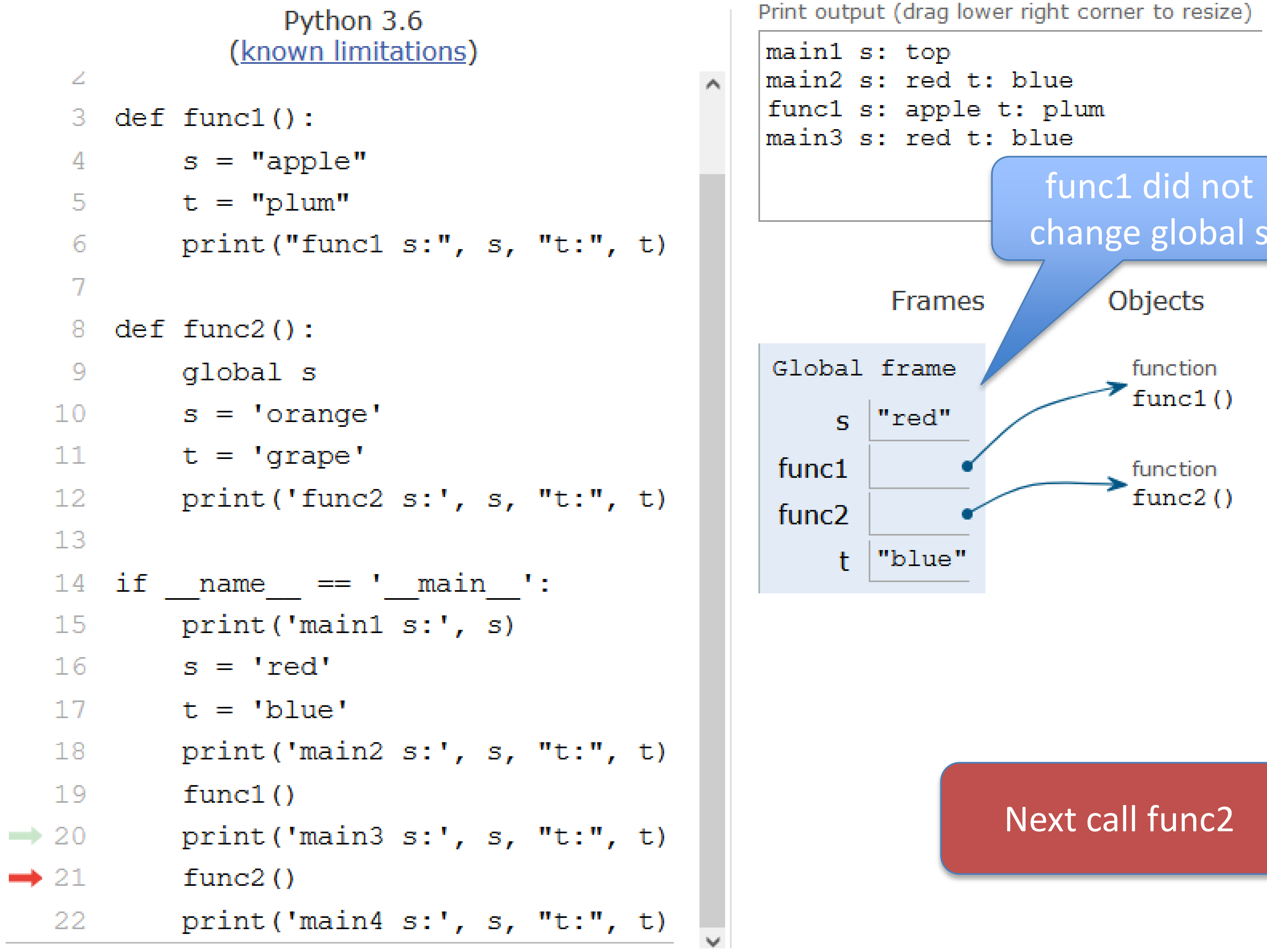
```
main1 s: top
main2 s: red t: blue
func1 s: apple t: plum
```

s is global

s is local variable

There are two different s variables

Python Tutor – Step 16



```
def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

```
main1 s: top
main2 s: red t: blue
func1 s: apple t: plum
main3 s: red t: blue
```

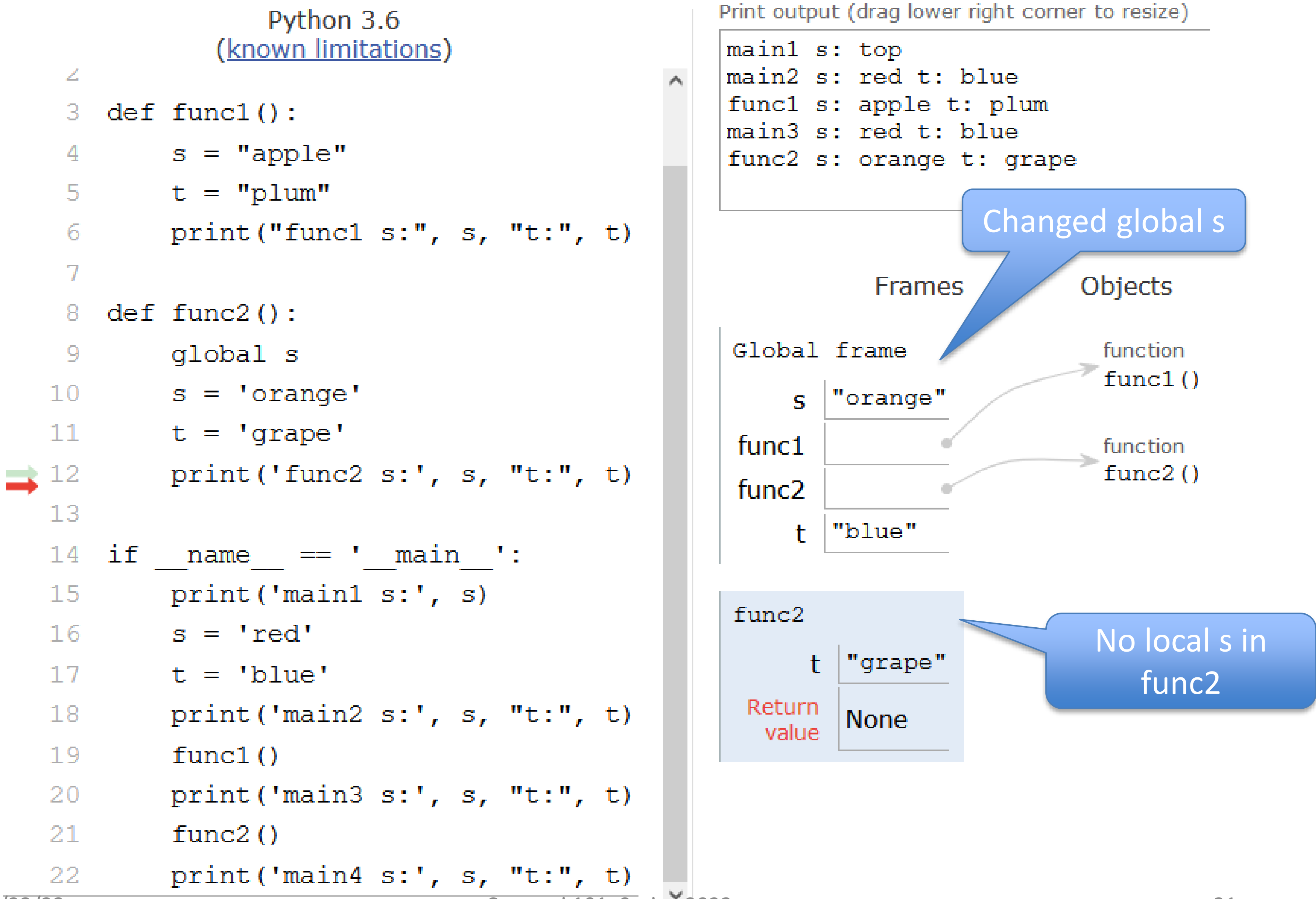
Global frame

s	"red"
func1	→ function func1()
func2	→ function func2()
t	"blue"

func1 did not change global s

Next call func2

Python Tutor – Step 21



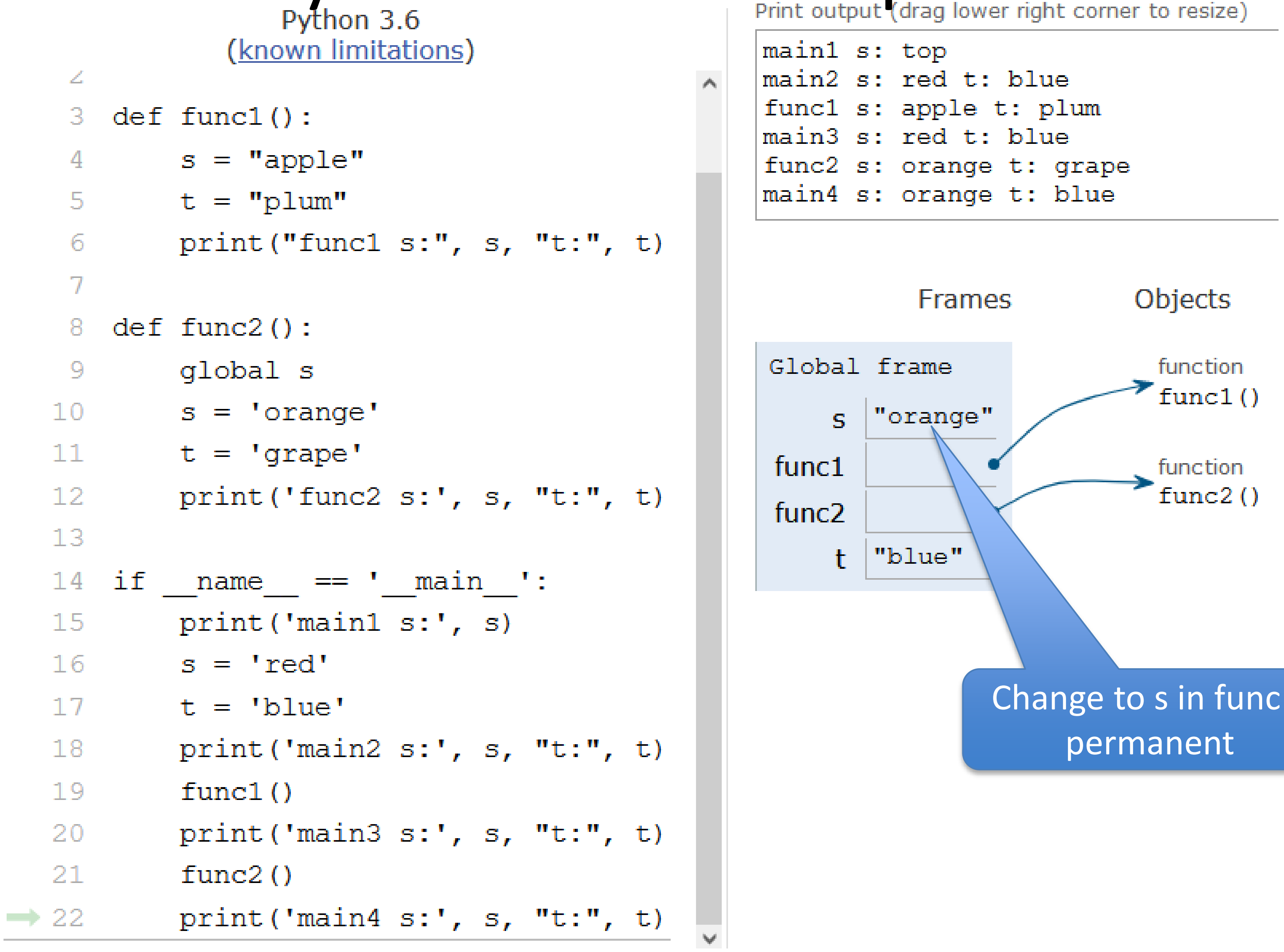
Python 3.6
(known limitations)

```
def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

Print output (drag lower right corner to resize)

```
main1 s: top
main2 s: red t: blue
func1 s: apple t: plum
main3 s: red t: blue
func2 s: orange t: grape
```

Python Tutor – Step 23



```python
def func1():
    s = "apple"
    t = "plum"
    print("func1 s:", s, "t:", t)

def func2():
    global s
    s = 'orange'
    t = 'grape'
    print('func2 s:', s, "t:", t)

if __name__ == '__main__':
    print('main1 s:', s)
    s = 'red'
    t = 'blue'
    print('main2 s:', s, "t:", t)
    func1()
    print('main3 s:', s, "t:", t)
    func2()
    print('main4 s:', s, "t:", t)
```

```
main1 s: top
main2 s: red t: blue
func1 s: apple t: plum
main3 s: red t: blue
func2 s: orange t: grape
main4 s: orange t: blue
```

Change to s in func 2 permanent

Variables
What, where, read, write? (in 101)

What is it?	Where first created?	Where accessible? (read)	Where reassign-able? (write)
Regular variable in main	In main	In main only (technically anywhere, but don’t do that)	In main only
Regular local function variable	In function	In function only	In function only
Global variable	Top of file	If not reassigning the value, in main and all functions	In main or in any function that first declares it global

Variables
What, where, read, write? (in 101)

What is it?	Where first created?	Where accessible? (read)	Where reassign-able? (write)
Regular variable in main	In main	In main only (technically anywhere, but don't do that)	In main only
Regular local function variable	In function	In function only	In function only
Global variable	Top of file	If not reassigning the value, in main and all functions	In main or in any function that first declares it global

Python will have an error if it is not declared global and it is used and then there is a variable with the same name being assigned

Can avoid this by ALWAYS declaring the variable global in the function (best practice) if that is the variable you are using

Assignment 3 Transform

- **Uses several global variables.**

- **Only use global variables when we specify in an assignment**

WOTO-1 – Tuples and Globals http://bit.ly/101s23-0223-1

Tuples

```
t = ([1], 2, ‘three’)
t[1] = 3
print(t[0][0])
print(type(t[0][0]))
t[0][0] = 4
print(t)
(x, y)=(t[1], t[0][0])
print(x,y)
```

```
t = ([1], 2, 'three')
t[1] = 3
print(t[0][0])
print(type(t[0][0]))
t[0][0] = 4
print(t)
(x, y)=(t[1], t[0][0])
print(x,y)                          2 4
print((x,y))                        (2,4)
```

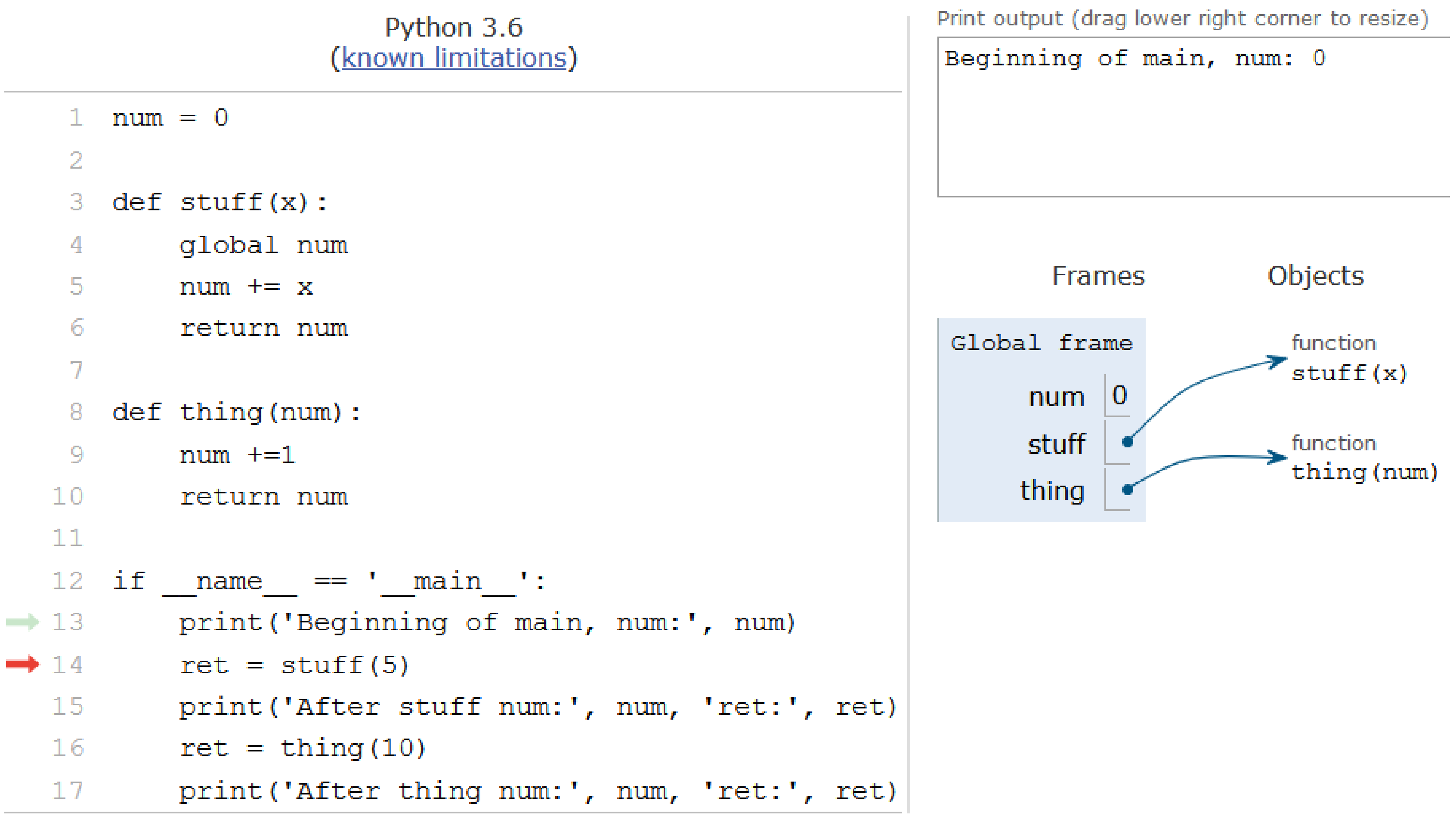
Notice there is NO variable assigned. There is no z = (x,y). This is a way to assign two variables at the same time. We are creating x and y both on the same line as new variables and giving them values

Tuples

```
t = ([1], 2, 'three')
t[1] = 3                              ERROR!!!B
print(t[0][0])                        1
print(type(t[0][0]))                  <class 'int'>
t[0][0] = 4
print(t)                              ([4],2,'three')
(x, y)=(t[1], t[0][0])
print(x,y)                            2 4
print((x,y))                          (2,4)
x = t[1]
y = t[0][0]
print(x,y)                            2 4
```

Similar!

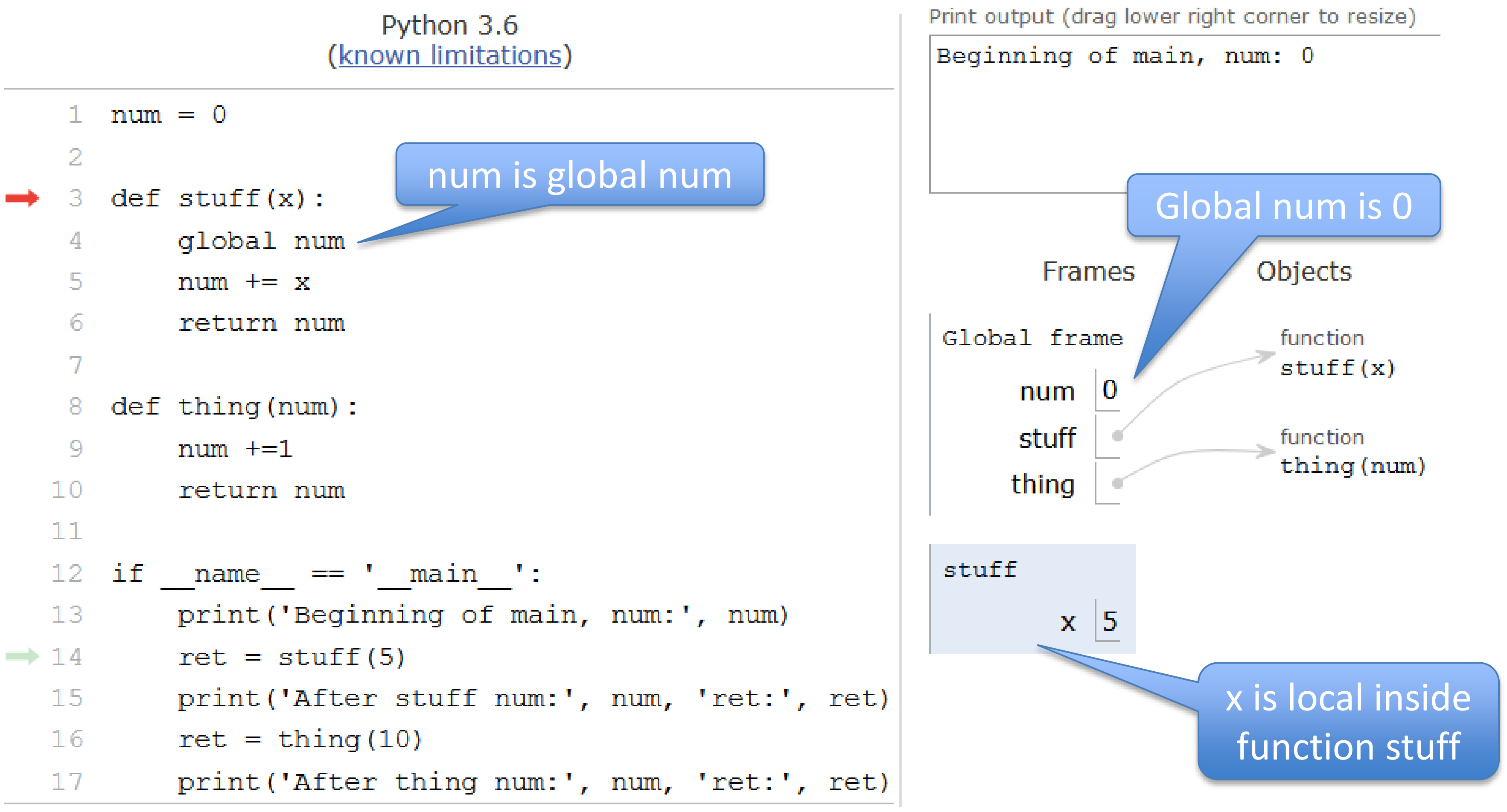
WOTO step through – step 6



```
num = 0

def stuff(x):
    global num
    num += x
    return num

def thing(num):
    num +=1
    return num

if __name__ == '__main__':
    print('Beginning of main, num:', num)
    ret = stuff(5)
    print('After stuff num:', num, 'ret:', ret)
    ret = thing(10)
    print('After thing num:', num, 'ret:', ret)
```

```
Beginning of main, num: 0
```

WOTO step through – step 7



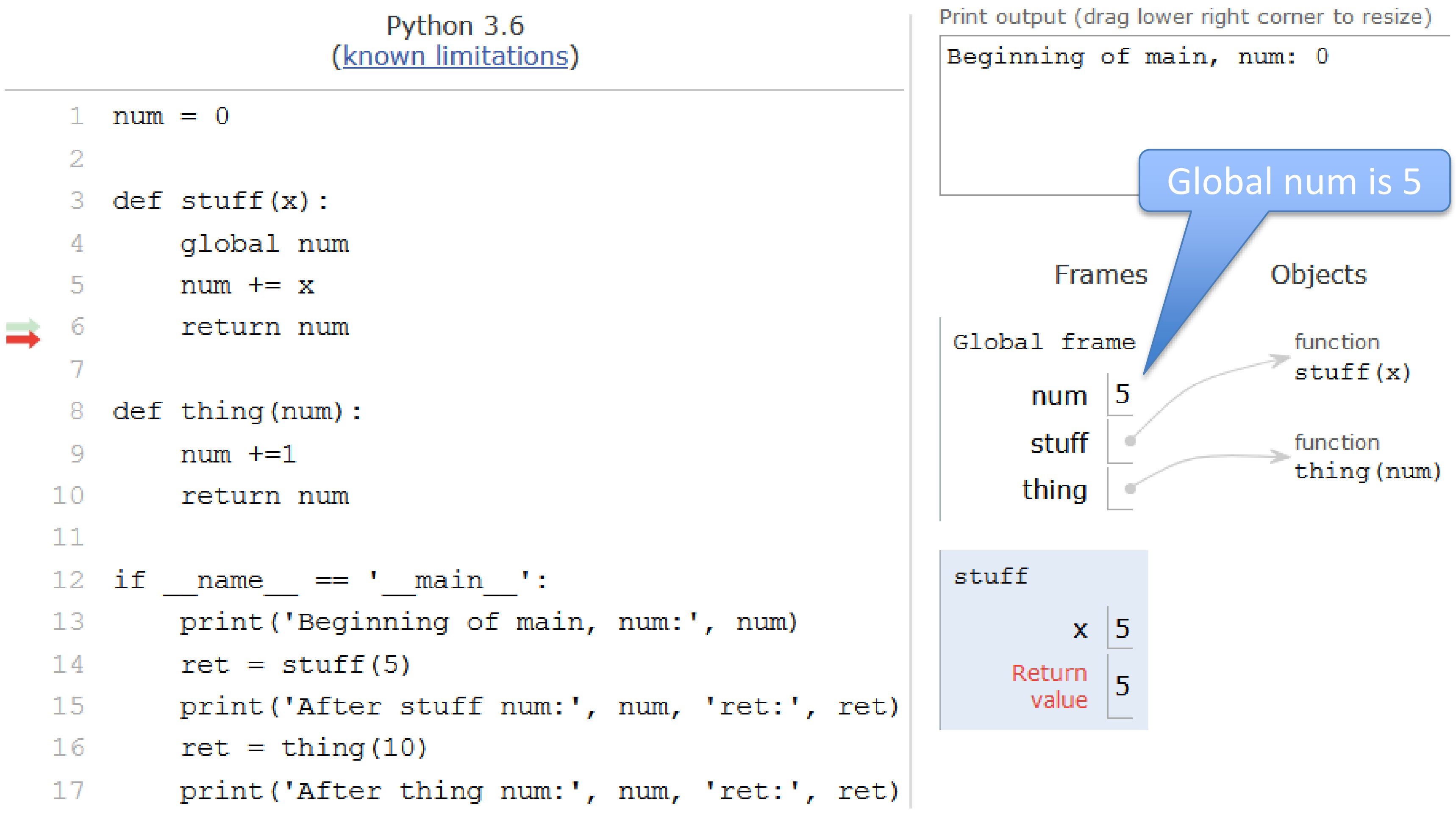
Python 3.6
(known limitations)

```
num = 0

def stuff(x):
    global num
    num += x
    return num

def thing(num):
    num +=1
    return num

if __name__ == '__main__':
    print('Beginning of main, num:', num)
    ret = stuff(5)
    print('After stuff num:', num, 'ret:', ret)
    ret = thing(10)
    print('After thing num:', num, 'ret:', ret)
```

Print output (drag lower right corner to resize)

```
Beginning of main, num: 0
```

WOTO step through – step 10



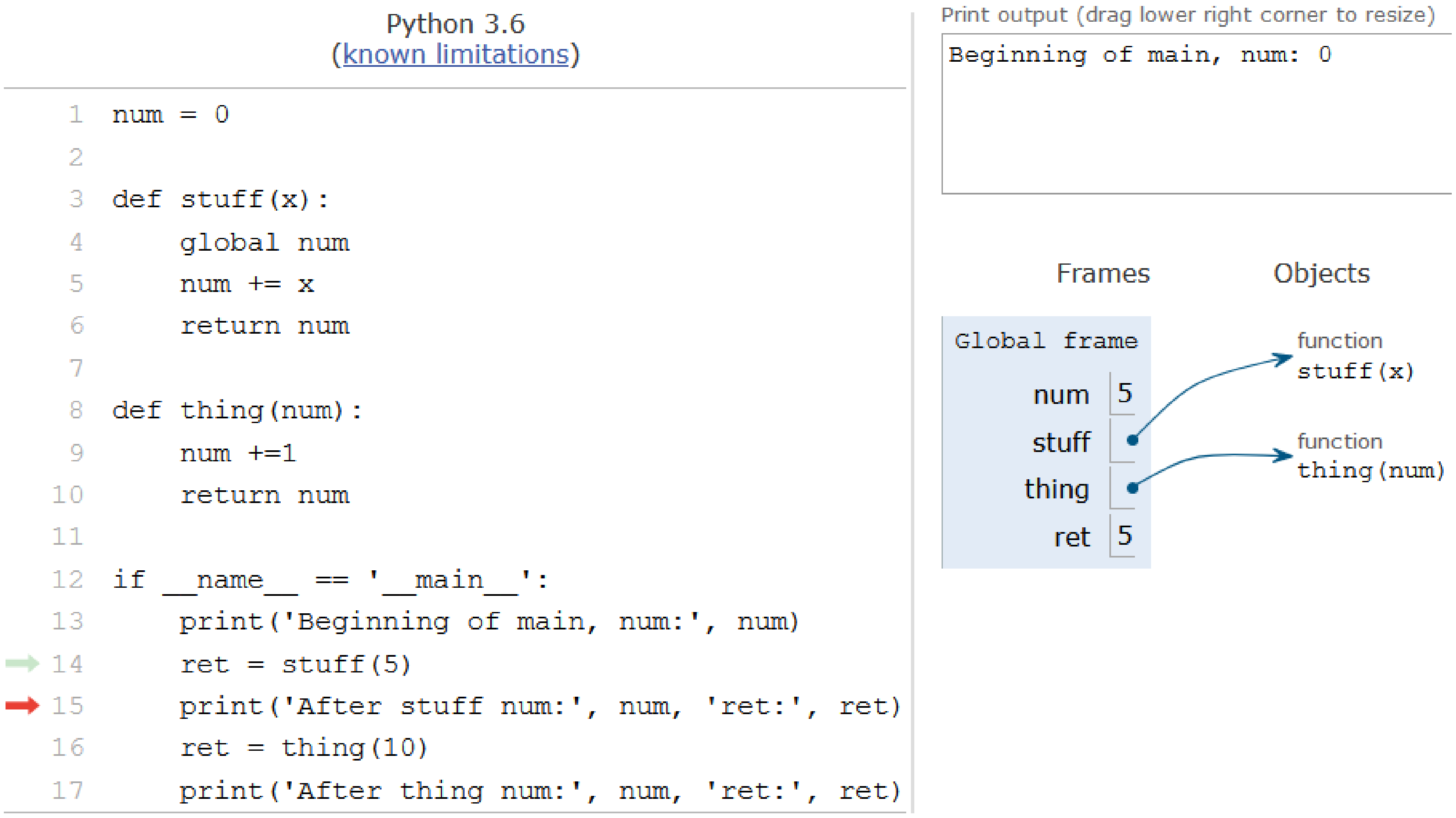
Python 3.6
(known limitations)

```
num = 0

def stuff(x):
    global num
    num += x
    return num

def thing(num):
    num +=1
    return num

if __name__ == '__main__':
    print('Beginning of main, num:', num)
    ret = stuff(5)
    print('After stuff num:', num, 'ret:', ret)
    ret = thing(10)
    print('After thing num:', num, 'ret:', ret)
```

Print output (drag lower right corner to resize)

```
Beginning of main, num:  0
```

WOTO step through – step 11



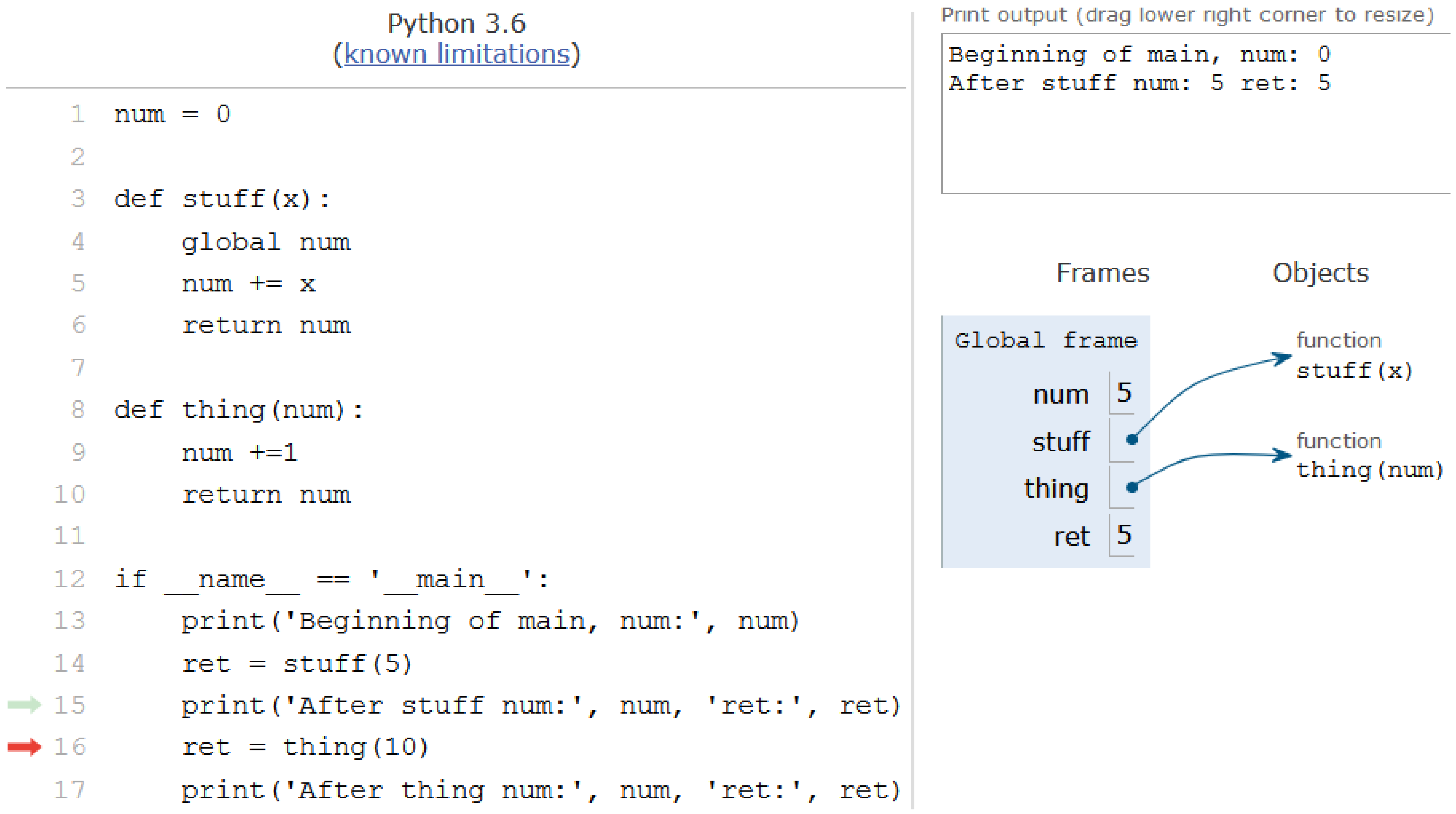
Python 3.6
(known limitations)

```python
num = 0

def stuff(x):
    global num
    num += x
    return num

def thing(num):
    num +=1
    return num

if __name__ == '__main__':
    print('Beginning of main, num:', num)
    ret = stuff(5)
    print('After stuff num:', num, 'ret:', ret)
    ret = thing(10)
    print('After thing num:', num, 'ret:', ret)
```

Print output (drag lower right corner to resize)

```
Beginning of main, num: 0
```

WOTO step through – step 12



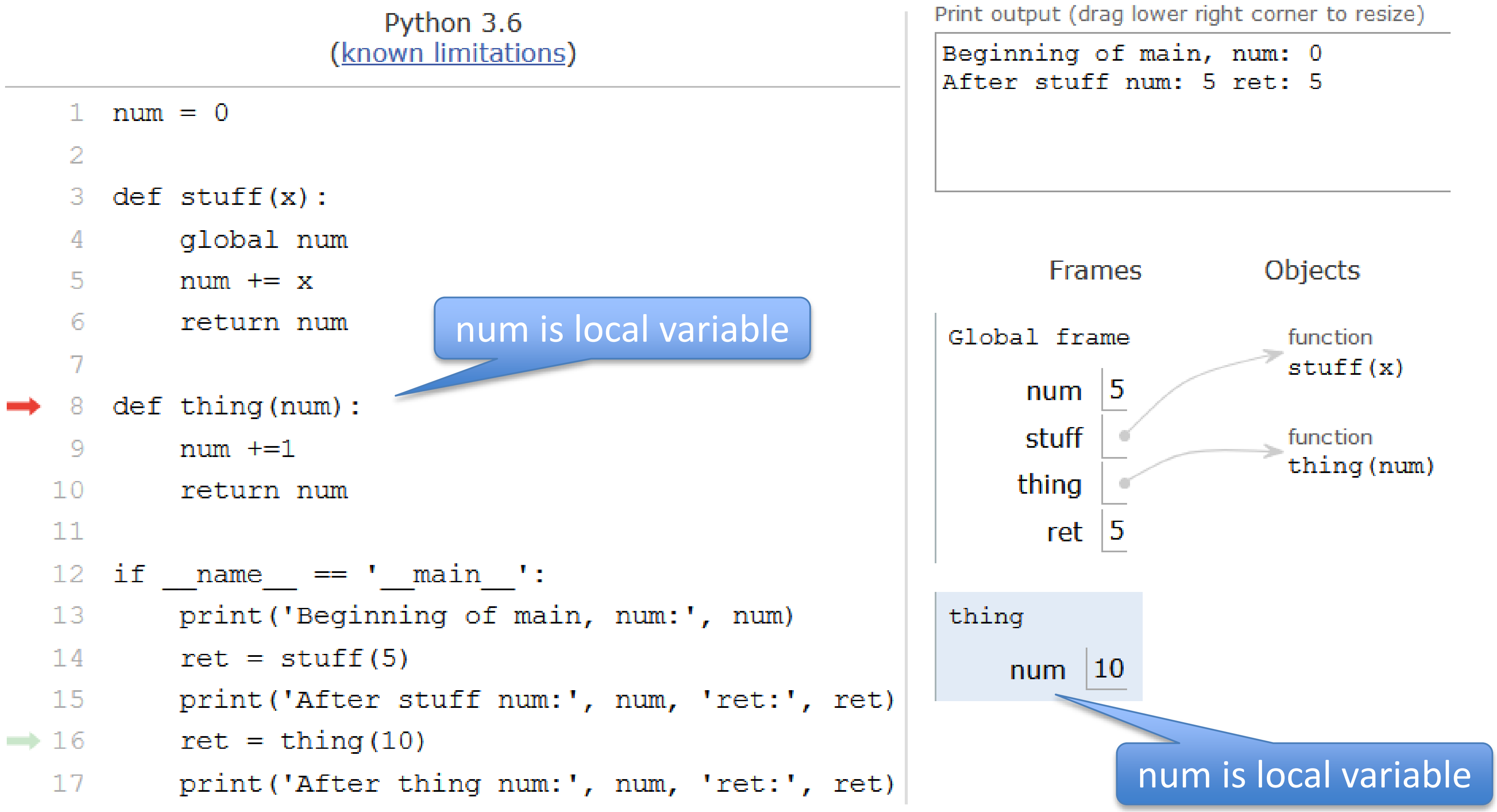
Python 3.6
(known limitations)

```
num = 0

def stuff(x):
    global num
    num += x
    return num

def thing(num):
    num +=1
    return num

if __name__ == '__main__':
    print('Beginning of main, num:', num)
    ret = stuff(5)
    print('After stuff num:', num, 'ret:', ret)
    ret = thing(10)
    print('After thing num:', num, 'ret:', ret)
```

Print output (drag lower right corner to resize)

```
Beginning of main, num: 0
After stuff num: 5 ret: 5
```

WOTO step through – step 13



```
num = 0

def stuff(x):
    global num
    num += x
    return num

def thing(num):
    num +=1
    return num

if __name__ == '__main__':
    print('Beginning of main, num:', num)
    ret = stuff(5)
    print('After stuff num:', num, 'ret:', ret)
    ret = thing(10)
    print('After thing num:', num, 'ret:', ret)
```

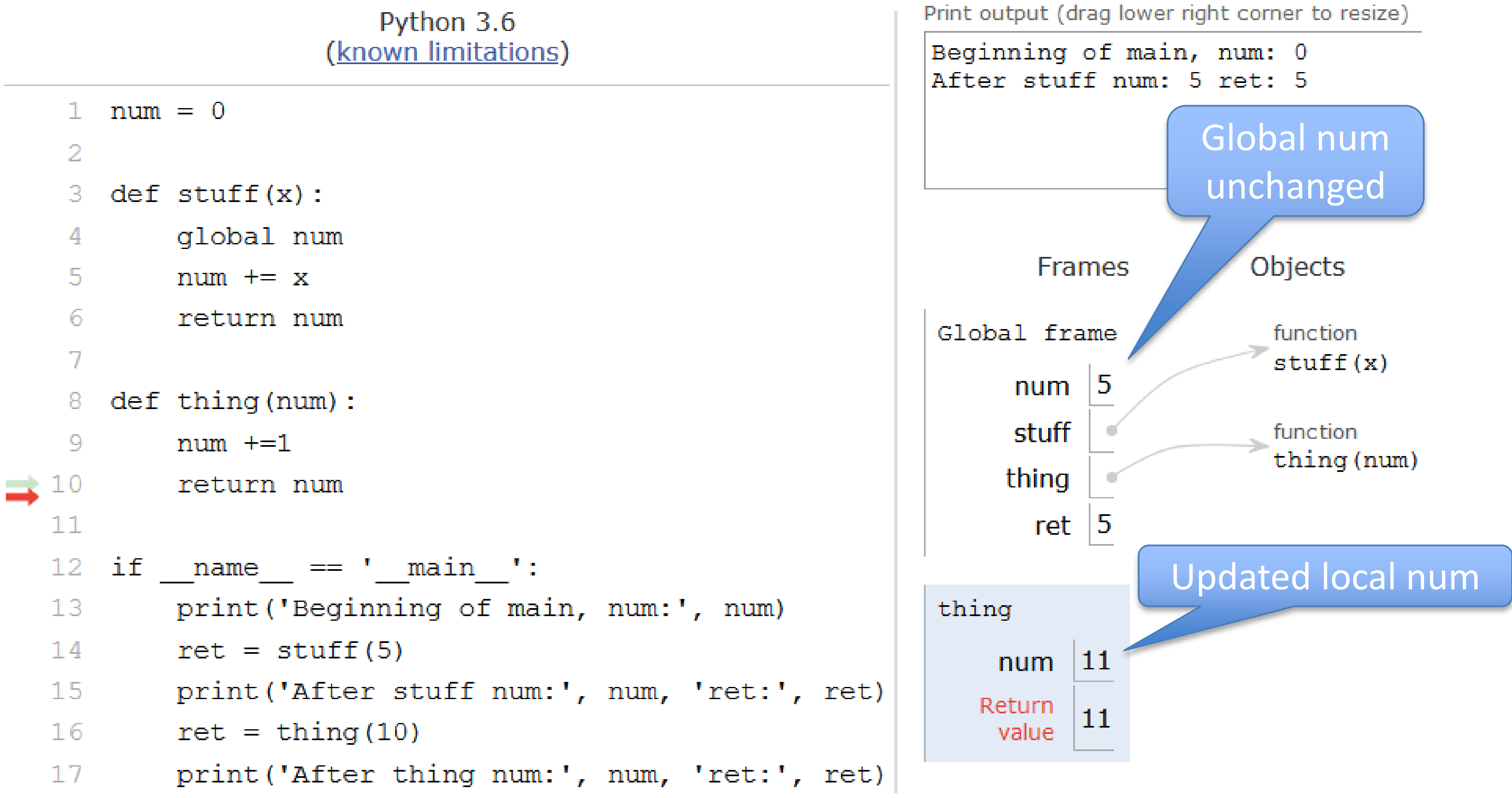
Print output (drag lower right corner to resize)

```
Beginning of main, num: 0
After stuff num: 5 ret: 5
```

num is local variable

num is local variable

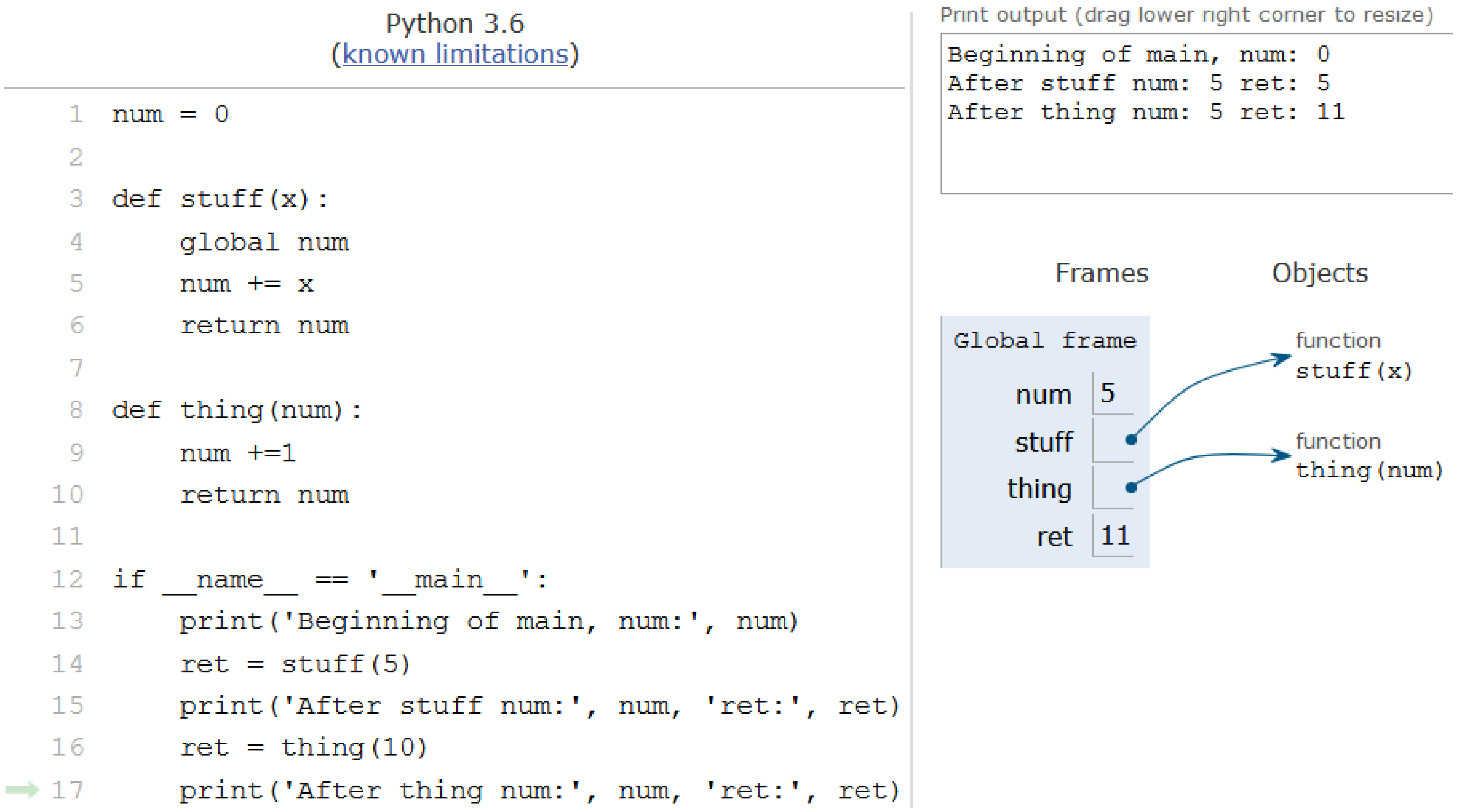
WOTO step through – step 16



```
num = 0

def stuff(x):
    global num
    num += x
    return num

def thing(num):
    num +=1
    return num

if __name__ == '__main__':
    print('Beginning of main, num:', num)
    ret = stuff(5)
    print('After stuff num:', num, 'ret:', ret)
    ret = thing(10)
    print('After thing num:', num, 'ret:', ret)
```

```
Beginning of main, num: 0
After stuff num: 5 ret: 5
```

WOTO step through – last step



```
num = 0

def stuff(x):
    global num
    num += x
    return num

def thing(num):
    num +=1
    return num

if __name__ == '__main__':
    print('Beginning of main, num:', num)
    ret = stuff(5)
    print('After stuff num:', num, 'ret:', ret)
    ret = thing(10)
    print('After thing num:', num, 'ret:', ret)
```

Print output (drag lower right corner to resize)

```
Beginning of main, num: 0
After stuff num: 5 ret: 5
After thing num: 5 ret: 11
```

List .index vs String .find

```
str = "computer“                    Values:
pos = str.find("m")
pos = str.find("b")

lst = ["a", "b", "c", "a"]
indx = lst.index("b")
indx = lst.index("B")
```

List .index vs String .find

```
str = "computer“                    Values:
pos = str.find("m")                 pos is 2
pos = str.find("b")                 pos is -1

lst = ["a", "b", "c", "a"]
indx = lst.index("b")               indx is 1
indx = lst.index("B")               ERROR, crash!
```

lst.index(item)   program crashes if item is not there!

List .index vs String .find

```
str = "computer“
pos = str.find("m")
pos = str.find("b")

lst = ["a", "b", "c", "a"]
indx = lst.index("b")
indx = lst.index("B")

indx = -1
if "B" in lst:
    indx = lst.index("B")
```

Ask if "in" lst, before using .index

Values:

pos is 2

pos is -1

indx is 1

ERROR, crash!

Use .index this way

Check if in!

Let's Write list Index function

- **Call in findIndex(lst, elm)**
- **Write it so it works like the string find function**
  - **lst** is a list
  - **elm** is an element
  - Return the position of **elm** in **lst**
  - Return **-1** if **elm** not in **lst**
  - Use while loop to implement
- **What is the while loop's Boolean condition?**

```
index = 0
while BOOL_CONDITION:
   index += 1
```

While Boolean condition

```
index = 0
while BOOL_CONDITION:
   index += 1
```

- **What is the while loop’s stopping condition?**

While Boolean condition

```
index = 0
while BOOL_CONDITION:
   index += 1
```

- **What is the while loop’s stopping condition?**
  - Whether found value: `lst[index] == elm`
  - Whether reach end of list: `index >= len(lst)`

DeMorgan’s Law

- **While loop stopping conditions, stop with either:**

- **While loop needs negation: DeMorgan's Laws**

  `not (A and B)` equivalent to **`(not A) or (not B)`**

  `not (A or B)` equivalent to **`(not A) and (not B)`**

DeMorgan's Law

- **While loop stopping conditions, stop with either:**
  - `lst[index] == elm`
  - `index >= len(lst)`
- **While loop needs negation: DeMorgan's Laws**

  `not (A and B)` equivalent to **`(not A) or (not B)`**

  `not (A or B)` equivalent to **`(not A) and (not B)`**

DeMorgan's Law

- **While loop stopping conditions, stop with either:**
  - `lst[index] == elm`
  - `index >= len(lst)`
- **While loop needs negation: DeMorgan's Laws**

  `not (A and B)` equivalent to **`(not A) or (not B)`**

  → **`not (A or B)`** equivalent to **`(not A) and (not B)`**

```
while not (lst[index] == elm or index >= len(lst)):
```

Is equivalent to: (not A) and (not B)

```
while lst[index] != elm and index < len(lst):
```

DeMorgan's Law

- **While loop stopping conditions, stop with either:**
  - `lst[index] == elm`
  - `index >= len(lst)`
- **While loop needs negation: DeMorgan's Laws**

`not (A and B)` equivalent to **`(not A) or (not B)`**

→ **`not (A or B)`** equivalent to **`(not A) and (not B)`**

```
while not (lst[index] == elm or index >= len(lst)):
```

Why did == become != ?

```
while lst[index] != elm and index < len(lst):
```

DeMorgan's Law

- **While loop stopping conditions, stop with either:**
  - `lst[index] == elm`
  - `index >= len(lst)`
- **While loop needs negation: DeMorgan's Laws**

  `not (A and B)` equivalent to **`(not A) or (not B)`**

  `not (A or B)` equivalent to **`(not A) and (not B)`**

```
while not (lst[index] == elm or index >= len(lst)):
```

Why did >= become < ?

```
while lst[index] != elm and index < len(lst):
```

Think: DeMorgan's Law

Fill in the blanks

A	B	not (A and B)	(not A) or (not B)
True	True		False
True	False	True	
False	True		True
False	False	True	

A	B	not (A or B)	(not A) and (not B)
True	True	False	
True	False		False
False	True	False	
False	False		True

Fill in the blanks

Think: DeMorgan's Law

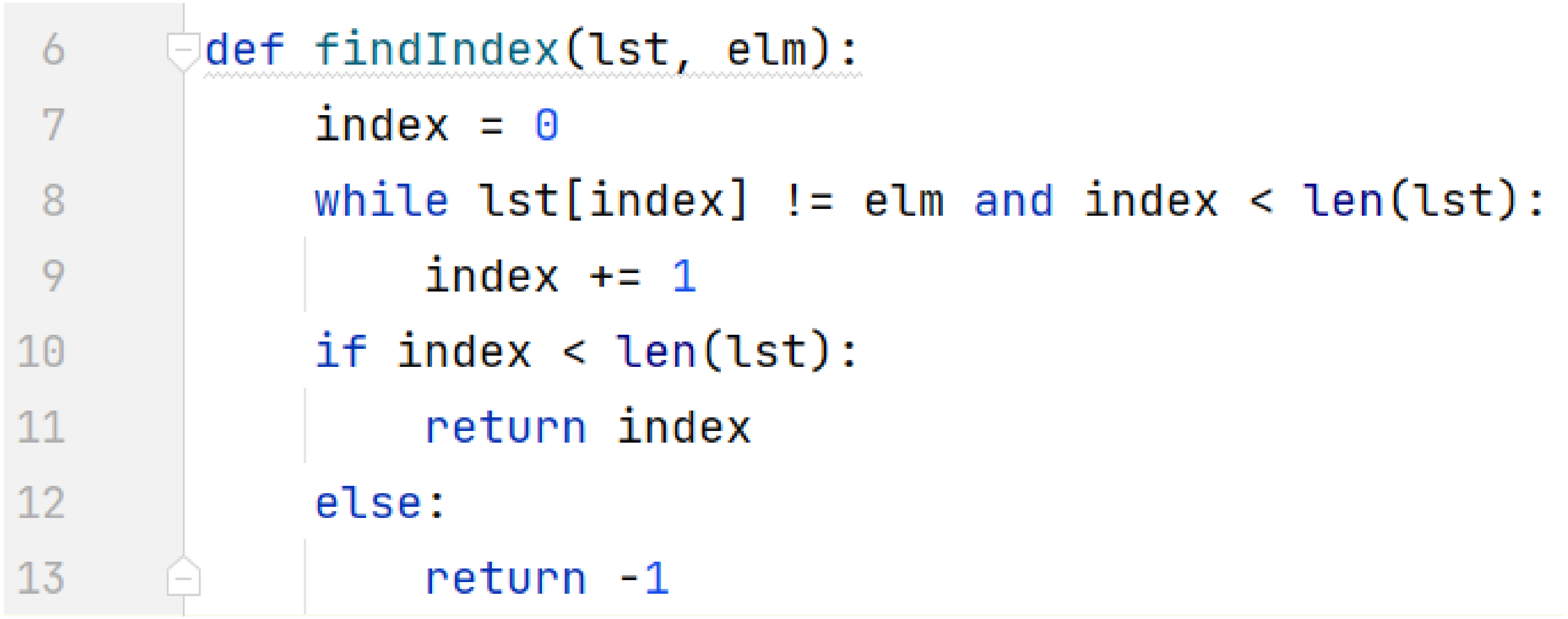
A	B	not (A and B)	(not A) or (not B)
True	True	False	False
True	False	True	True
False	True	True	True
False	False	True	True

A	B	not (A or B)	(not A) and (not B)
True	True	False	False
True	False	False	False
False	True	False	False
False	False	True	True

WOTO-2: Will this work?
http://bit.ly/101s23-0223-2

WOTO-2: Will this work?

http://bit.ly/101s23-0223-2



```python
def findIndex(lst, elm):
    index = 0
    while lst[index] != elm and index < len(lst):
        index += 1
    if index < len(lst):
        return index
    else:
        return -1
```

Short Circuit Evaluation

- **Short circuit evaluation, these are not the same!**

- **As soon as truthiness of expression known**
  - Stop evaluating
  - In `(A and B)`, if A is false, do not evaluate B

Short Circuit Evaluation

- **Short circuit evaluation, these are not the same!**

```
while lst[index] != elm and index < len(lst):

while index < len(lst) and lst[index] != elm:
```

- **As soon as truthiness of expression known**
  - Stop evaluating
  - In (A and B), if A is false, do not evaluate B

Short Circuit Evaluation

- **Short circuit evaluation, these are not the same!**

First condition depends on second condition

```
while lst[index] != elm and index < len(lst):
```

Put second condition first!

```
while index < len(lst) and lst[index] != elm:
```

- **As soon as truthiness of expression known**
  - Stop evaluating
  - In (A and B), if A is false, do not evaluate B

Python Logic Summarized

- **A and B is True only when A is True and B is True**
  - If A is True
  - If A is False
- **A or B is True if one of A and B are True**
  - if A is True
  - If A is False

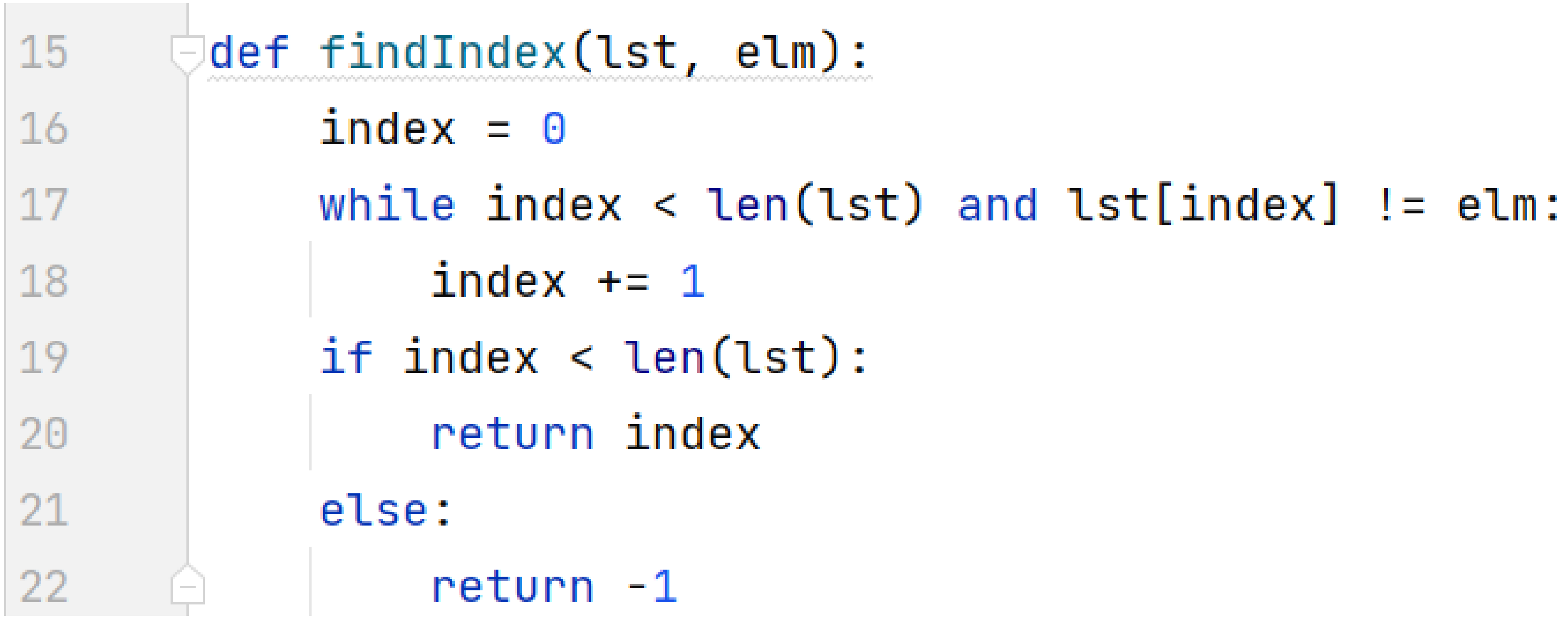
- **Short-circuit evaluation A and B, A or B**

Python Logic Summarized

- **A and B is True only when A is True and B is True**
  - If A is True — **Need to evaluate B**
  - If A is False — **Don't need to evaluate B**
- **A or B is True if one of A and B are True**
  - if A is True — **Don't need to evaluate B**
  - If A is False — **Need to evaluate B**

- **Short-circuit evaluation A and B, A or B**
  - If evaluation of A gives you the answer, you don't need to evaluate B

Correct Code:



```python
def findIndex(lst, elm):
    index = 0
    while index < len(lst) and lst[index] != elm:
        index += 1
    if index < len(lst):
        return index
    else:
        return -1
```

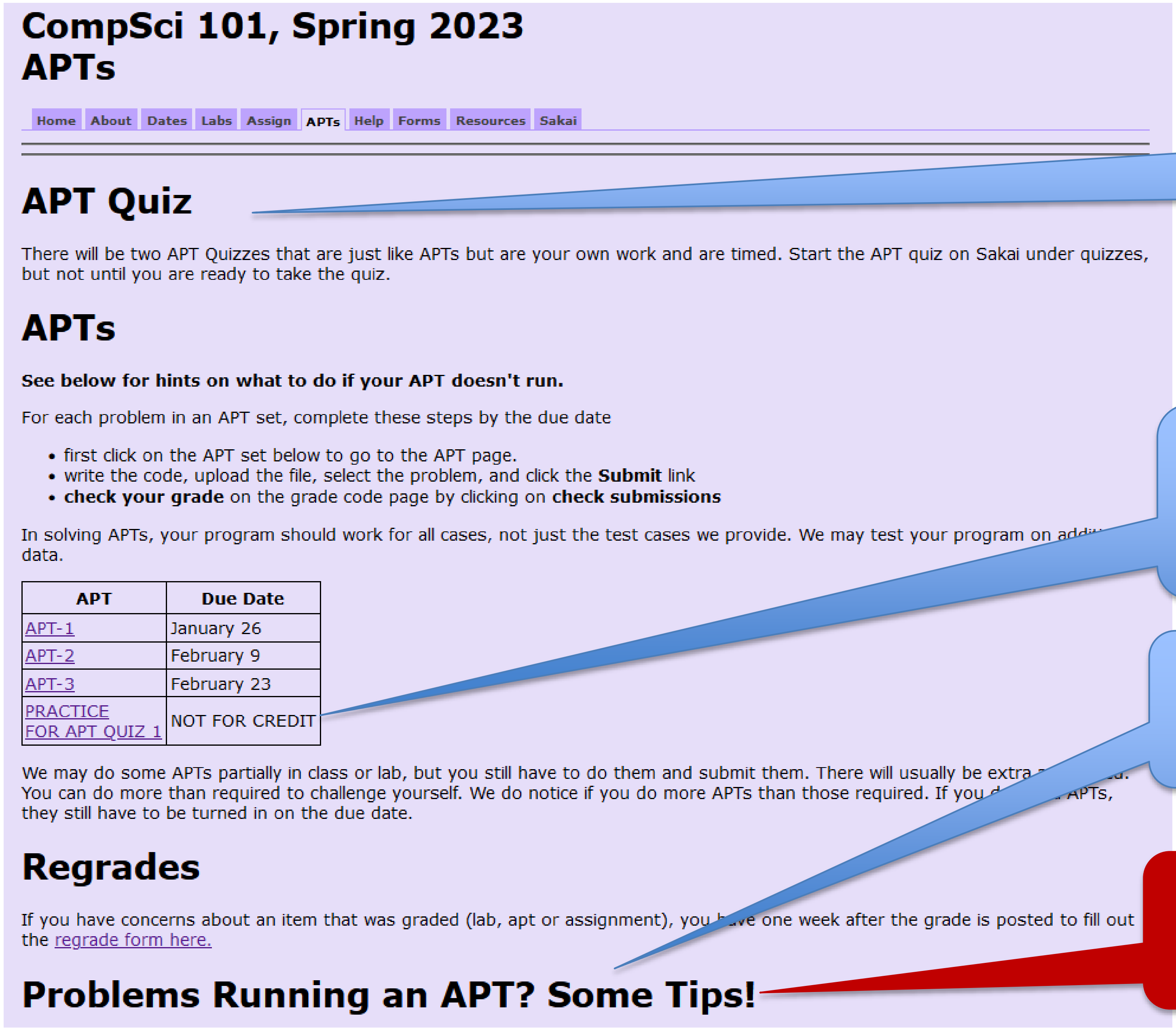
APT Quiz 1 Feb 23-27

- **Opens 2/23 1pm**
- **Closes at 11pm 2/27 – must finish all by this time**
- **There are two parts based on APTs 1-3**
  - Each part has two APT problems
  - Each part is 2 hours – more if you get accommodations
  - Each part starts in Sakai under tests and quizzes
  - Sakai is a starting point with countdown timer that sends you to a new apt page just for each part
  - Could do each part on different day or same days
- **Old APT Quiz so you can practice (not for credit) – on APT Page**

APT Quiz 1

We take cheating seriously in this course!

- **Is your own work!**
  - No collaboration with others!
  - Use your notes, lecture notes, your code, textbook
  - DO NOT search for answers!
  - Do not talk to others about the quiz until grades are posted
- **Post private questions on Ed Discussion**
  - We are not online between 9pm and 9am!
  - We are not on all the time, especially weekends
  - Will try to answer questions between 9am – 9pm
    - About typos, cannot help you in solving APTs
- **See 101 APT page for tips on debugging APTs**



CompSci 101, Spring 2023
APTs

Home | About | Dates | Labs | Assign | APTs | Help | Forms | Resources | Sakai

APT Quiz

There will be two APT Quizzes that are just like APTs but are your own work and are timed. Start the APT quiz on Sakai under quizzes, but not until you are ready to take the quiz.

APTs

See below for hints on what to do if your APT doesn't run.

For each problem in an APT set, complete these steps by the due date

- first click on the APT set below to go to the APT page.
- write the code, upload the file, select the problem, and click the **Submit** link
- **check your grade** on the grade code page by clicking on **check submissions**

In solving APTs, your program should work for all cases, not just the test cases we provide. We may test your program on addi data.

APT	Due Date
APT-1	January 26
APT-2	February 9
APT-3	February 23
PRACTICE FOR APT QUIZ 1	NOT FOR CREDIT

We may do some APTs partially in class or lab, but you still have to do them and submit them. There will usually be extra d. You can do more than required to challenge yourself. We do notice if you do more APTs than those required. If you d APTs, they still have to be turned in on the due date.

Regrades

If you have concerns about an item that was graded (lab, apt or assignment), you e one week after the grade is posted to fill out the regrade form here.

Problems Running an APT? Some Tips!

APT Quiz Info

Practice (old APT quiz)

Debugging Tips

Stuck! Use 7 steps!

Don't go to Sakai to start APT Quiz until you are ready to start

If you click on it, you start it!

Tips for APT Quiz

- **Don't like the format, convert it:**
- **dig = "458"   Is variable dig a number?**
- **Use 7 steps**

Tips for APT Quiz

- **Don't like the format, convert it:**
  - "lots of words" → ["lots", "of", "words"]
  - "6 3 9" → ['6', '3', '9'] → [6, 3, 9]
- **dig = "458"   Is variable dig a number?**
  - Is each letter in "0123456789"?
  - For ch in dig:
  -    if ch not in "0123456789"
  -          # not a digit!
- **Use 7 steps**
  - Work an example by hand
  - Code – what do you need? Loop over what? If?

Tips for APT Quiz

- **Write a helper function**

- **Break code into parts**

Tips for APT Quiz

- **Write a helper function**
  - What if had function to do X?
    - Test function before you use it
  - If you have a loop inside a loop
    - Instead put the inside loop in a function and call it
    - Simplifies your code
    - Easier to debug
- **Break code into parts**
  - Do one part at a time
  - Print values of variables for each part
  - You think it does one thing, You might be surprised

Problem 1

- **Write function addto. Given wordlist, a list of words and numlist, a list of integers, return a new list with a number from numlist in the same position attached to the end of each string. Repeat numbers from numlist in the same order if you need more numbers**
  - numlist = [3, 5, 6]
  - wordlist = ["on", "to", "a", "be", "some", "fa", "so"]
  - Result: ["on3", "to5", "a6", "be3", "some5", "fa6", "so3"]
  - def addto(wordlist, numlist):
- **How to solve:**

WOTO-3: function addto
http://bit.ly/101s23-0223-3

Problem 1

- **Write function addto. Given list of words and list of integers, return new list with one number to end of each string, repeat numbers if you need more numbers**
  - numlist = [3, 5, 6]
  - wordlist = ["on", "to", "a", "be", "some", "fa", "so"]
  - Result: ["on3", "to5", "a6", "be3", "some5", "fa6", "so3"]
- **How to solve:**
  - Loop through numlist multiple times – TRICKY!
  - Easier: create "new" numlist that is longer
    - Create   nlist is   [3, 5, 6, 3, 5, 6, 3, 5, 6]
    - Use a for loop to do this
    - OR: nlist = numlist*3

Let's solve!

- **Make list of numbers long enough**
- **Use indexing**
  - Index into wordlist and same position in numlist
- **Use a loop over wordlist and create a new list**
  - Accumulation pattern!

Practice for APT Quiz 1

def addto(wordlist, numlist):

Practice for APT Quiz 1

```
def addto(wordlist, numlist):
    nlist = numlist
    answer = [ ]
    if len(numlist) < len(wordlist):
        nlist = numlist * len(wordlist)    # plenty big
    for index in range(len(wordlist)):
        answer.append(wordlist[index] + str(nlist[index]))
    return answer
```

Practice for APT Quiz 1

Index loop!

```
def addto(wordlist, numlist):
    nlist = numlist
    answer = [ ]
    if len(numlist) < len(wordlist):
        nlist = numlist * len(wordlist)    # plenty big
    for index in range(len(wordlist)):
        answer.append(wordlist[index] + str(nlist[index]))
    return answer
```

Create index variable, goes from 0 to size of list minus one

Use index

Problem 2

- **Write function update that has one parameter, a list of integers and/or words.**
- **This function makes a new list by starting with the original list and adds 1 to each number in the list. The string returned is the sum of the modified numbers in the list, a colon, followed by the elements in the modified list, separated by a dash**
- **Example:**
  - update([1, 5, 'a', 2, 'z']) returns "11:2-6-a-3-z"
  - update([87, 'car', 11, 'be']) returns "100:88-car-12-be"

How to solve

How to solve

- **For each element in list, is it a number?**
- **For numbers only add 1**
- **Sum only numbers, avoid strings**
- **Convert numbers to strings to build final string**

def update(alist):

```
def update(alist):
    onemore = [ ]
    for x in alist:
        if str(x)[0] in "0123456789":  # just check 1st digit
            onemore.append(x+1)   # add 1 to number
        else:
            onemore.append(x)      # add word
    total = 0
    for x in onemore:
        if str(x)[0] in "0123456789":  # if it is a number
            total += x
    final = [str(x) for x in onemore]  # convert all to strings
    return str(total)  +  ":"  +  "-".join(final)
```